Computer Graphics

CSC 418/2504

Patricio Simari

October 5, 2011

Slides courtesy of Patrick Coleman

Today

3D Transformations
Rendering Overview
3D Viewing

3D Affine Transformations

$$\mathbf{f}(\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{t}$$

3 x 3 Linear Transformation

3D Translation

3D Homogeneous Coordinates

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \equiv \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix}$$

Homogeneous Cartesian

3D Homogeneous Coordinate Examples

Homogeneous Cartesian Homogeneous Cartesian

$$\begin{bmatrix}3\\4\\5\\1\end{bmatrix} \equiv \begin{bmatrix}3\\4\\5\end{bmatrix} \qquad \begin{bmatrix}22\\33\\12\\11\end{bmatrix} \equiv \begin{bmatrix}2\\3\\12/11\end{bmatrix}$$

$$\begin{bmatrix}3\\4\\5\\4\end{bmatrix} \equiv \begin{bmatrix}3/4\\1\\5/4\end{bmatrix} \qquad \begin{bmatrix}3/4\\4/3\\1/6\\1/12\end{bmatrix} \equiv \begin{bmatrix}9\\16\\2\end{bmatrix}$$

3D Affine Transformations

Cartesian

$$\mathbf{f}(\mathbf{p}) = \mathbf{A}\mathbf{p} + \mathbf{t}$$

Homogeneous

$$\mathbf{f}(\mathbf{p}) = \mathbf{M}\mathbf{p}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} & & \\ & \mathbf{A} & \\ & & \end{bmatrix} & \begin{bmatrix} \\ \mathbf{t} \\ \\ \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

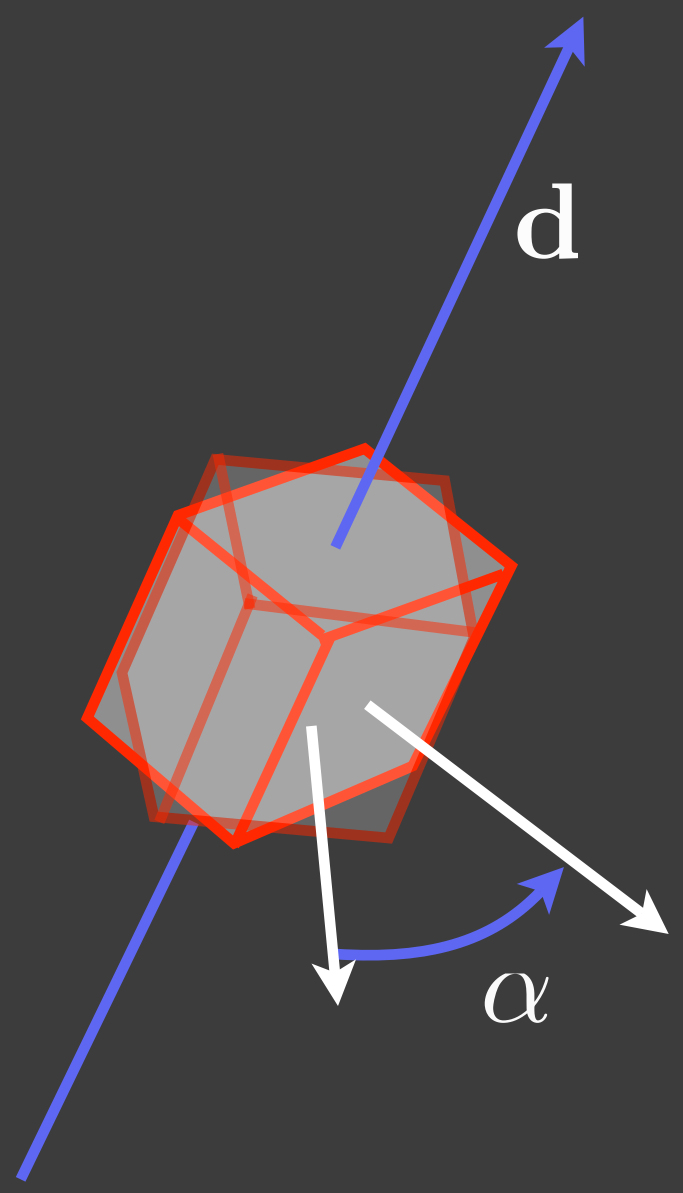
Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} & & \\ & \mathbf{I} & \\ & & \end{bmatrix} & \begin{bmatrix} \\ \mathbf{t} \\ \\ \end{bmatrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scaling

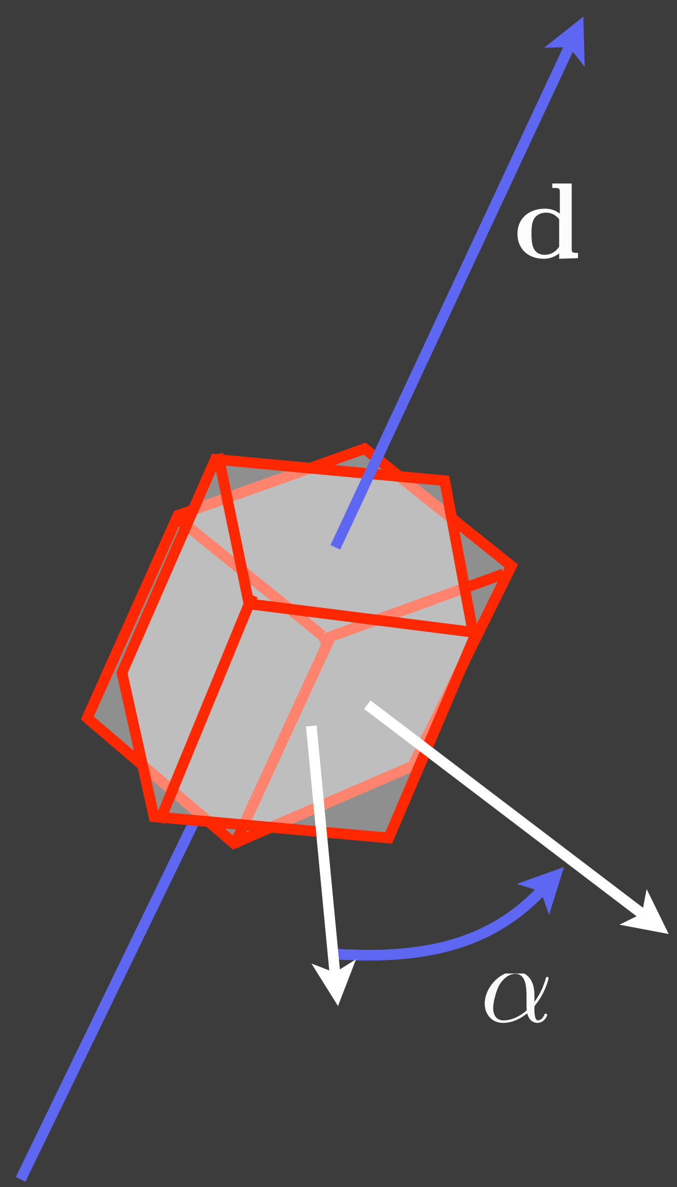
$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

3D Rotation



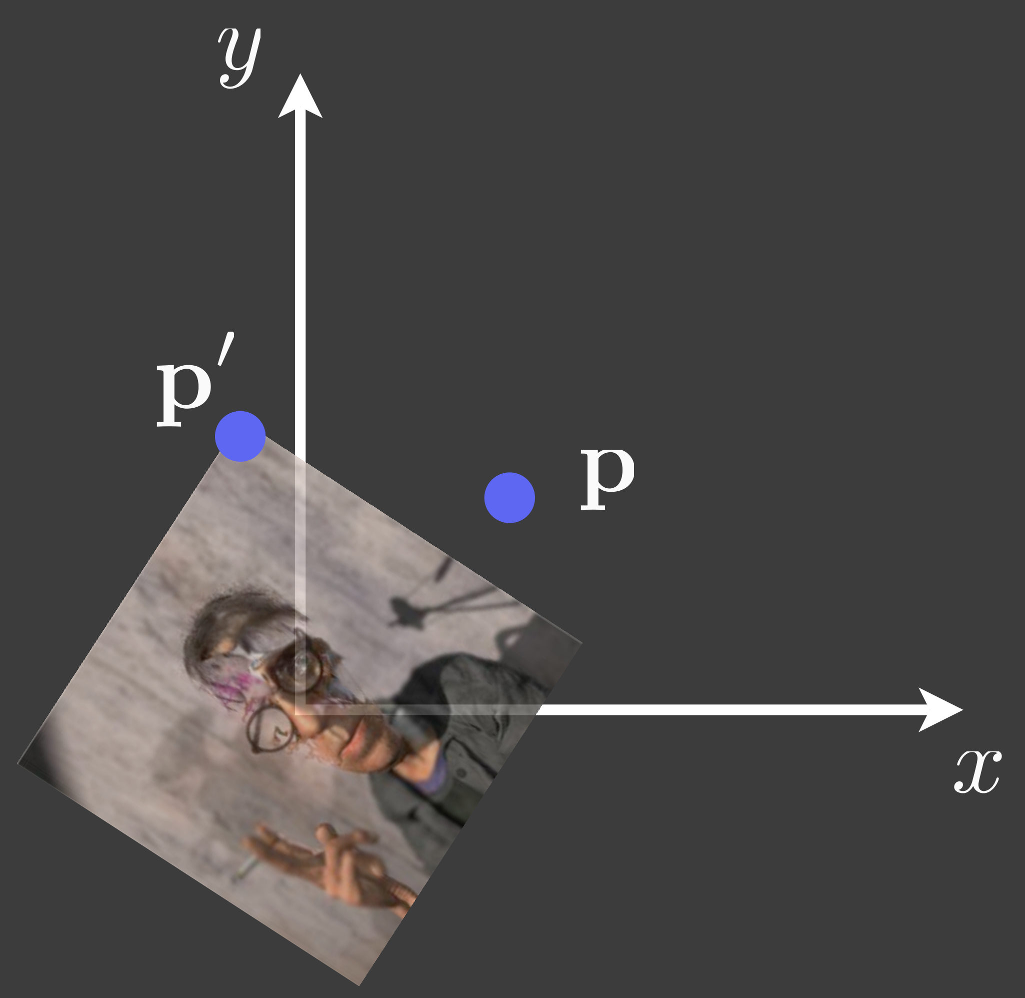
Any change in 3D orientation can be expressed as a rotation about some axis $\mathbf{d}$ by an angle α

3D Rotation



What is the affine transformation?

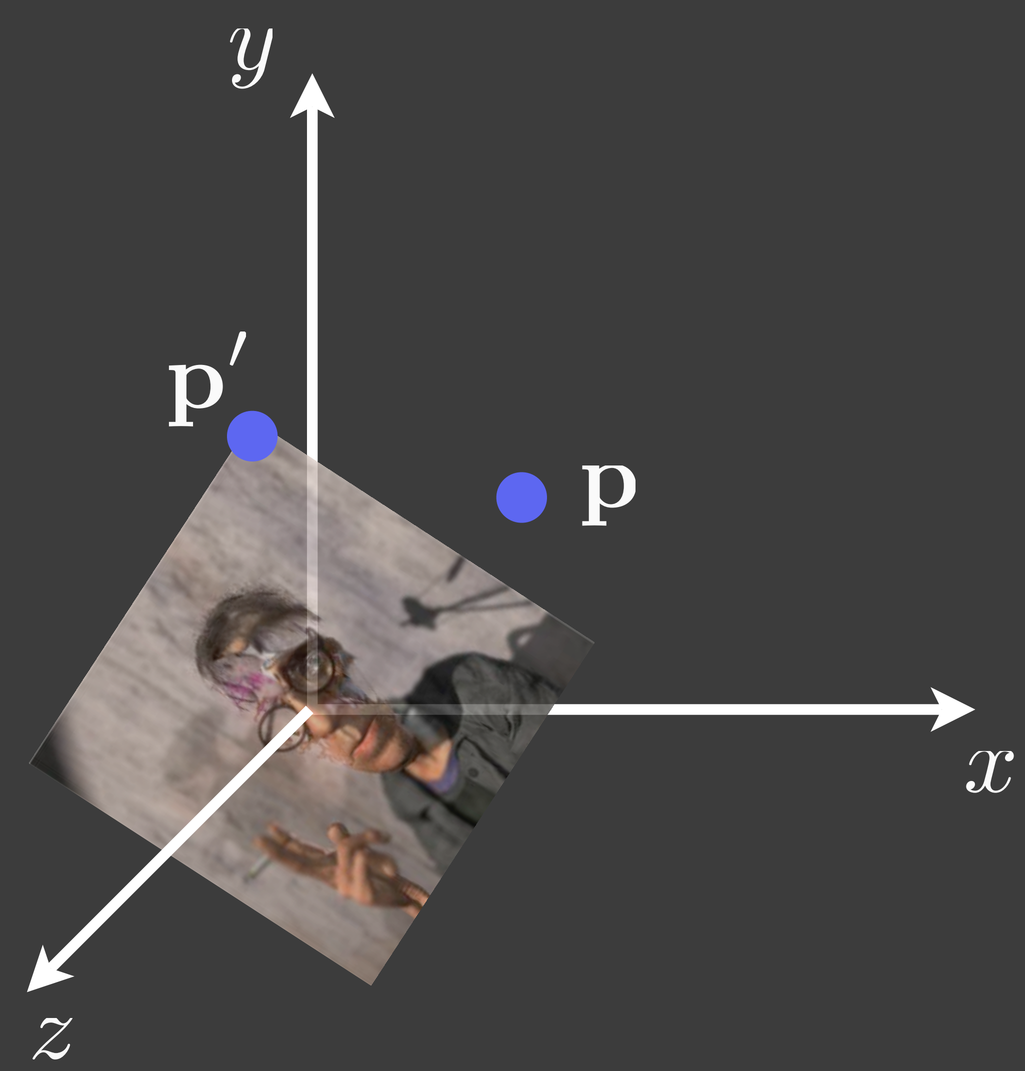
Recall 2D Rotation



$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Add an Axis

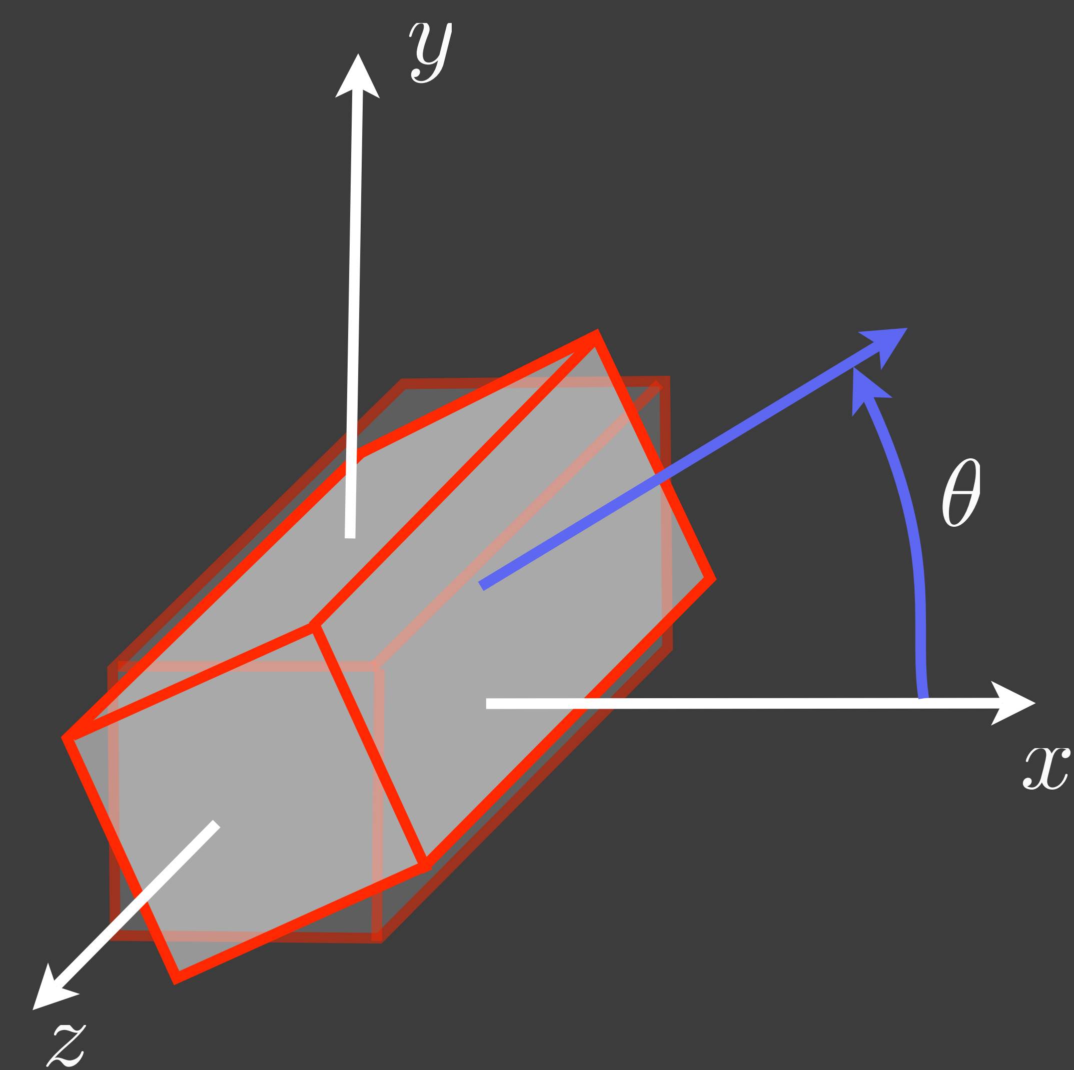


$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$
$$z' = z$$

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotation about z

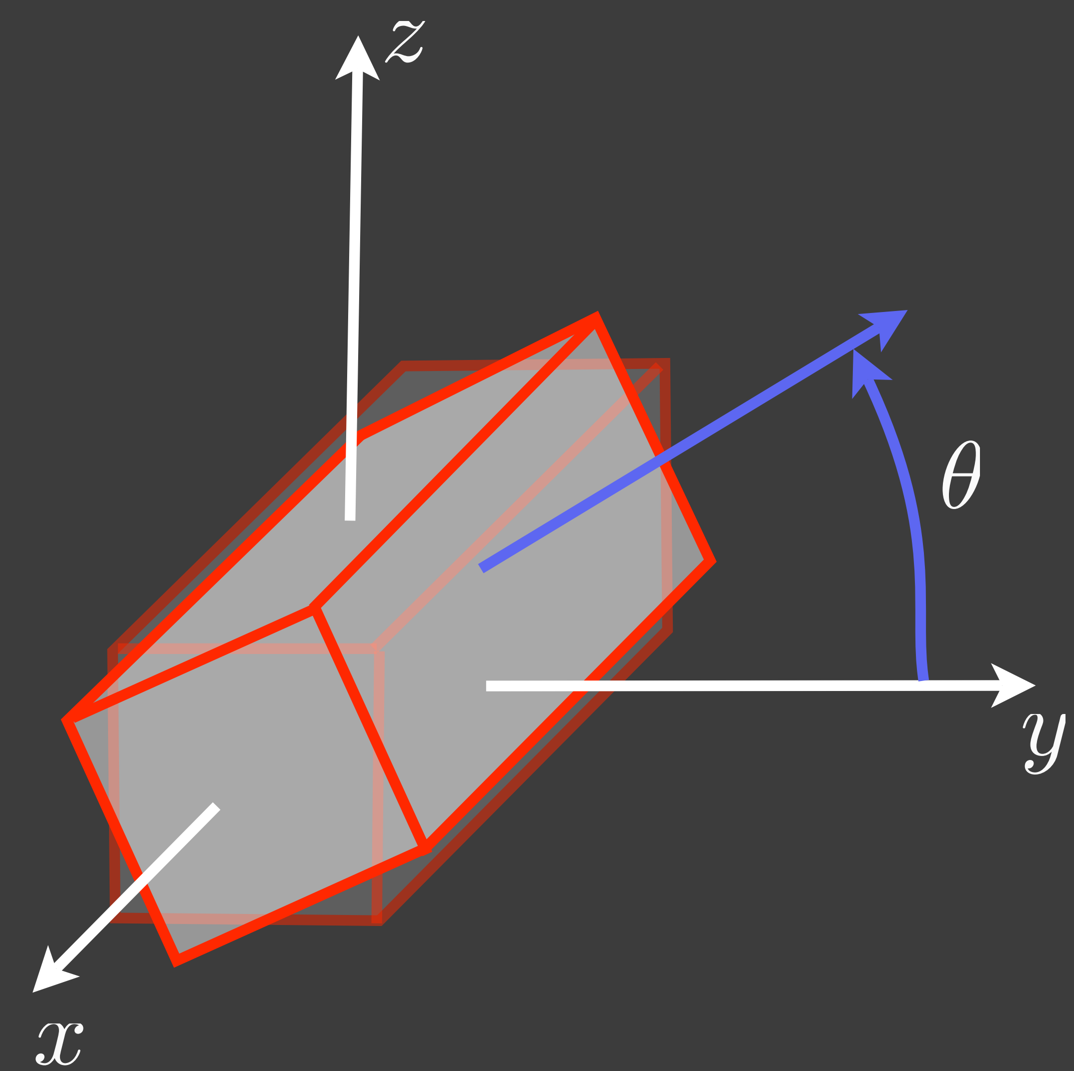
$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$
$$z' = z$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \underbrace{\begin{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{matrix} 0 & \qquad 0 & \qquad 0 \end{matrix} & 1 \end{bmatrix}}_{R_z(\theta)} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotation about x

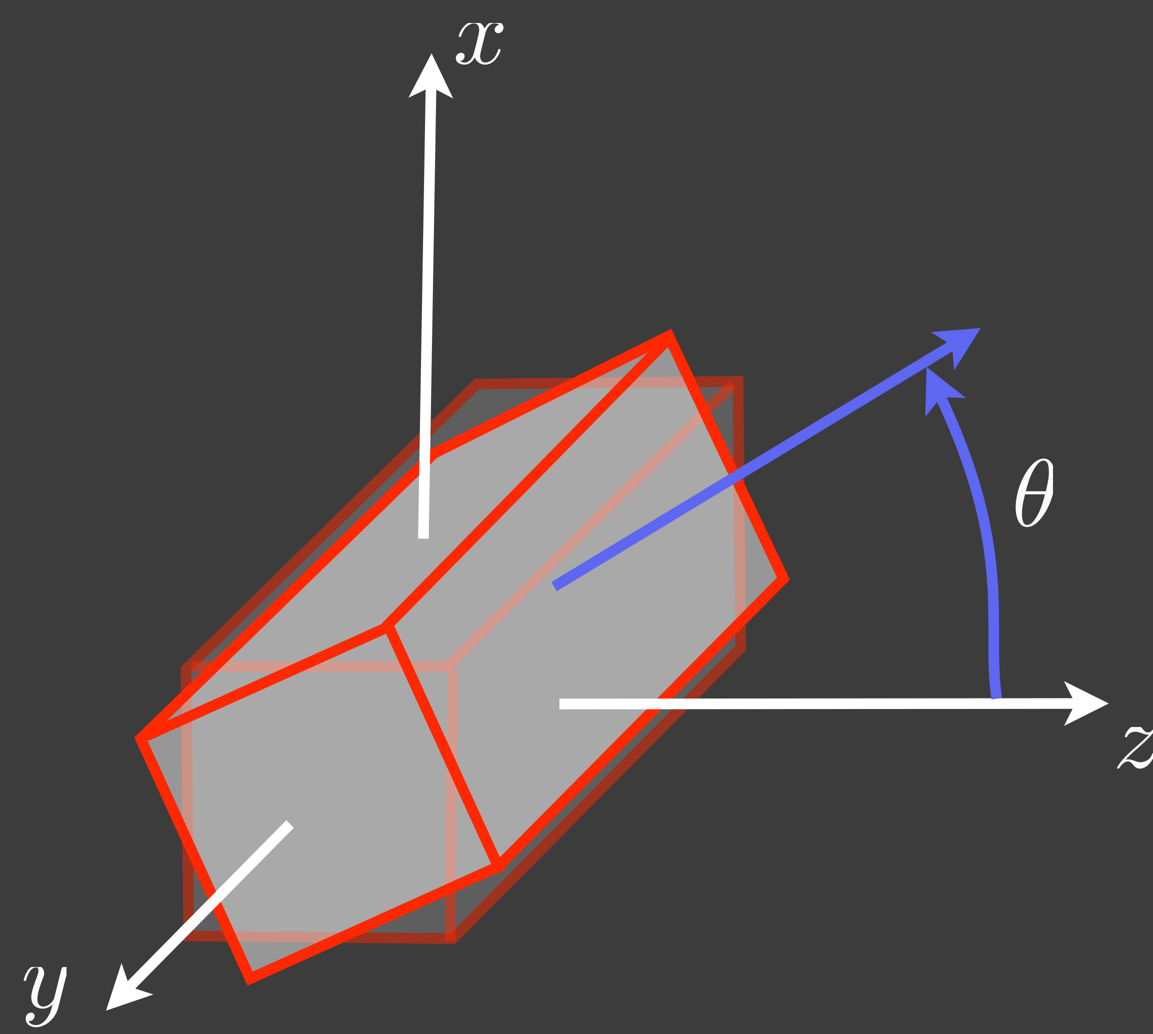
$$x' = x$$
$$y' = y\cos(\theta) - z\sin(\theta)$$
$$z' = y\sin(\theta) + z\cos(\theta)$$



$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \underbrace{\begin{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix}}_{R_x(\theta)} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotation about y

$$x' = x\cos(\theta) + z\sin(\theta)$$
$$y' = y$$
$$z' = -x\sin(\theta) + z\cos(\theta)$$

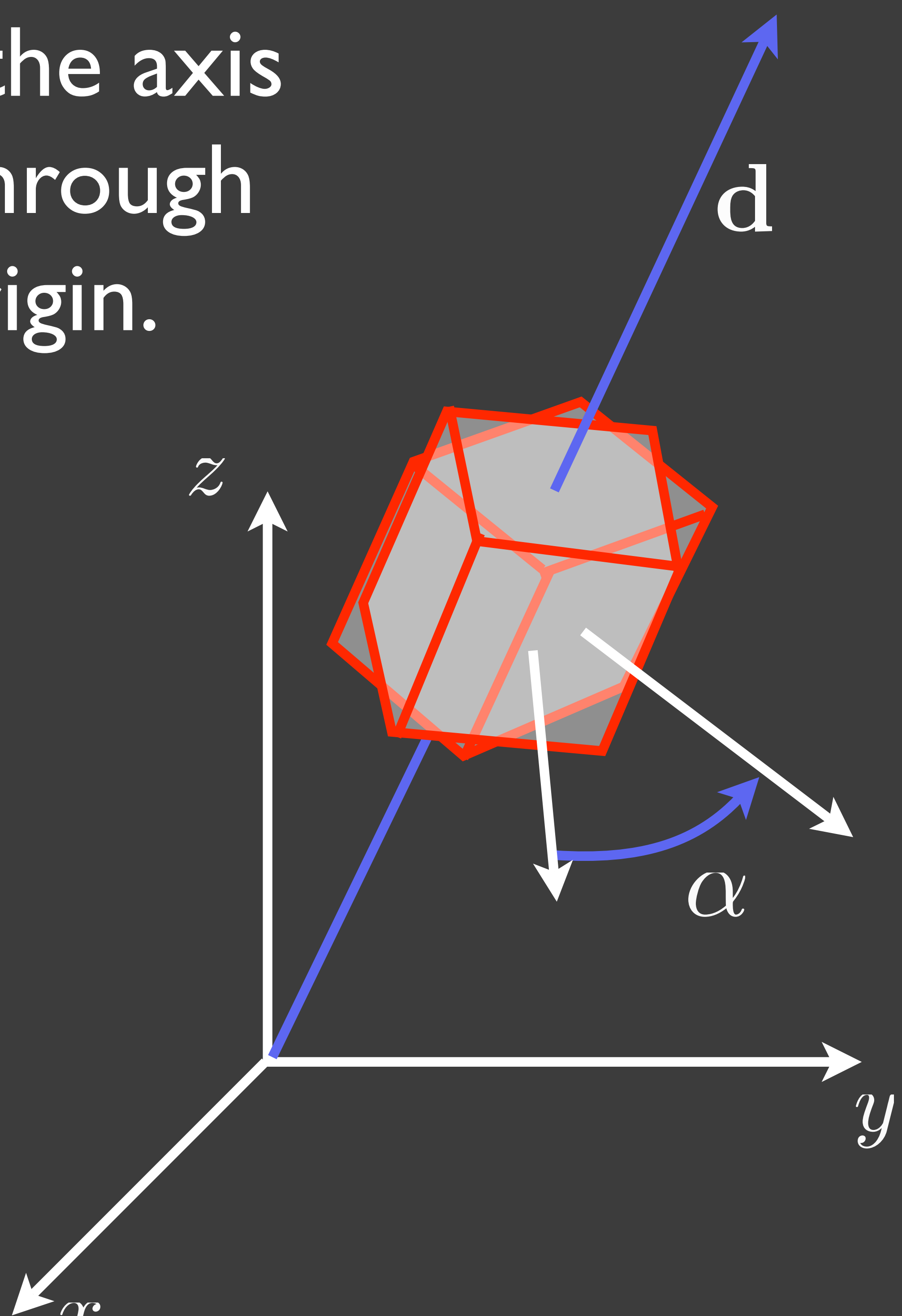


$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{matrix} 0 & 0 & 0 \end{matrix} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$R_y(\theta)$

Back to Rotation About an Axis

Assume the axis passes through the origin.

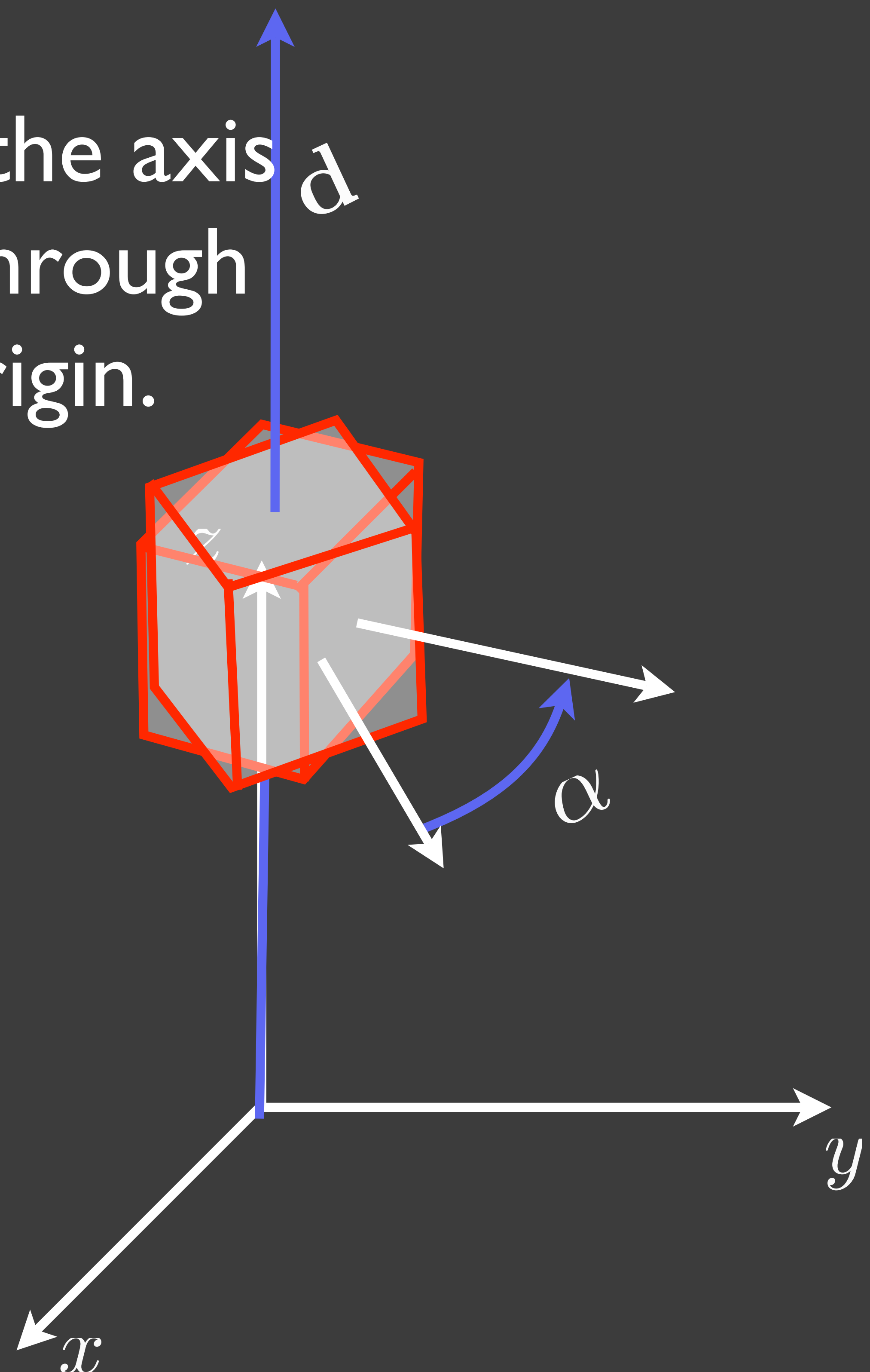


We know how to rotate about the x, y, and z axes individually.

1) Align d to x, y, or z
2) Rotate around it by α
3) Undo the Alignment

Back to Rotation About an Axis

Assume the axis d passes through the origin.

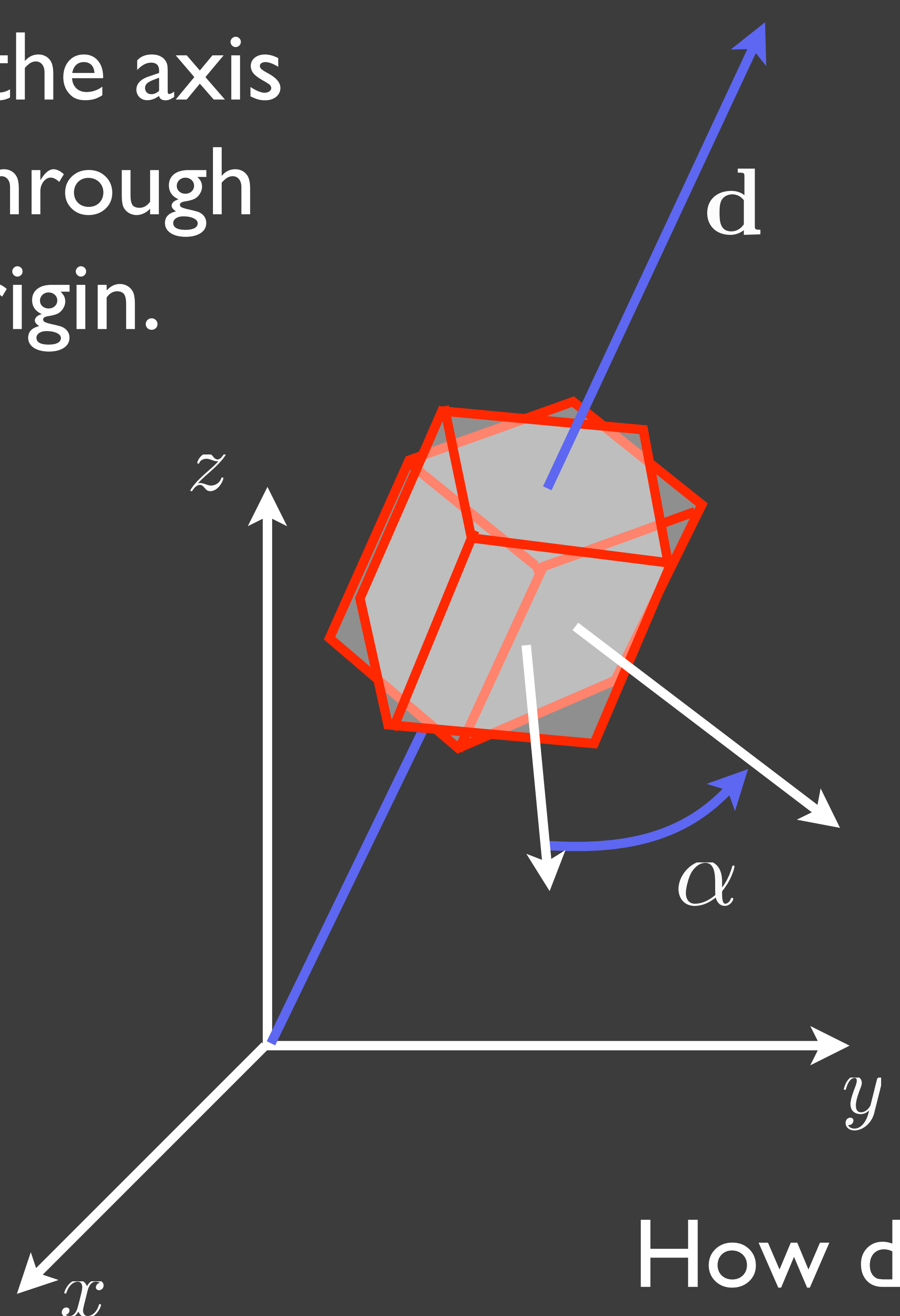


We know how to rotate about the x, y, and z axes individually.

1) Align d to x, y, or z
2) Rotate around it by α
3) Undo the Alignment

Back to Rotation About an Axis

Assume the axis passes through the origin.

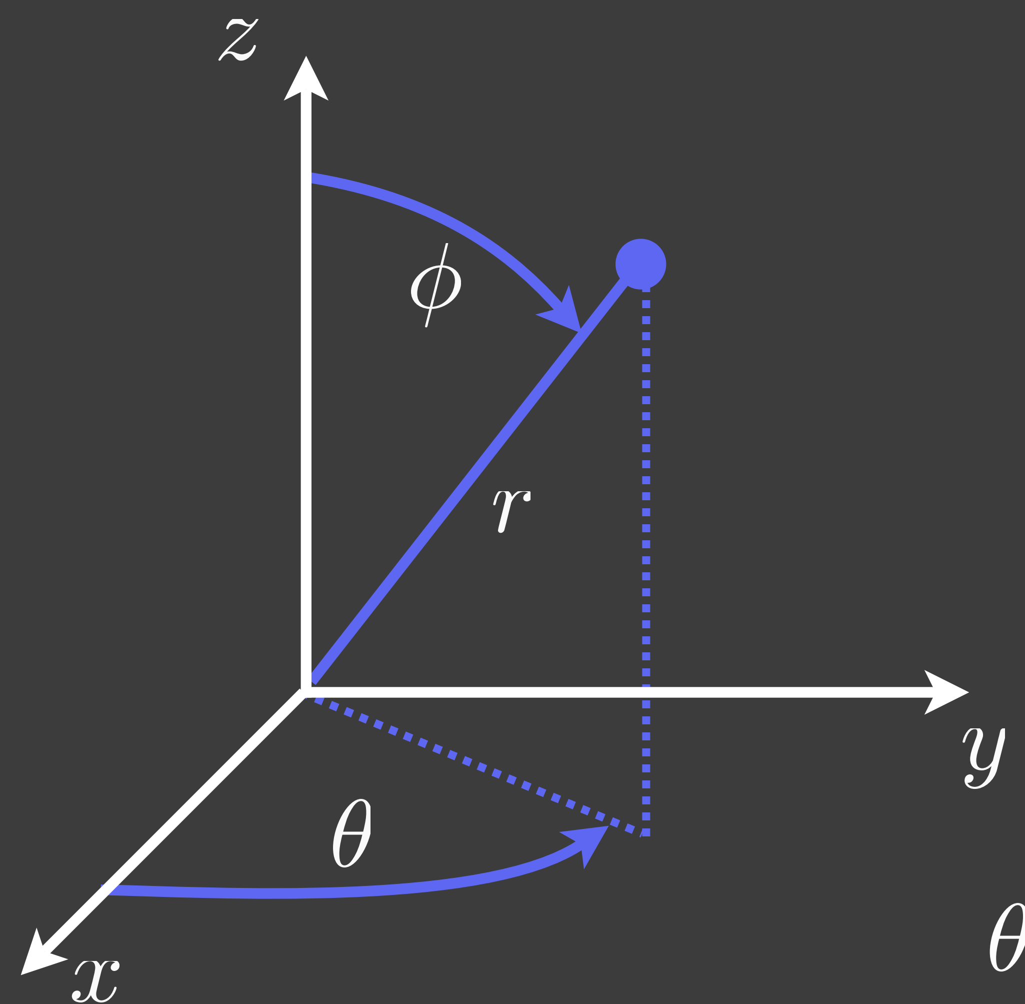


We know how to rotate about the x, y, and z axes individually.

1) Align $\mathbf{d}$ to x, y, or z
2) Rotate around it by α
3) Undo the Alignment

How do we align $\mathbf{d}$ to an axis?

Aside: Spherical Coordinates Review



$$\theta = \tan^{-1}\left(\frac{y}{x}\right)$$

$$x = \cos(\theta)\sin(\phi)$$

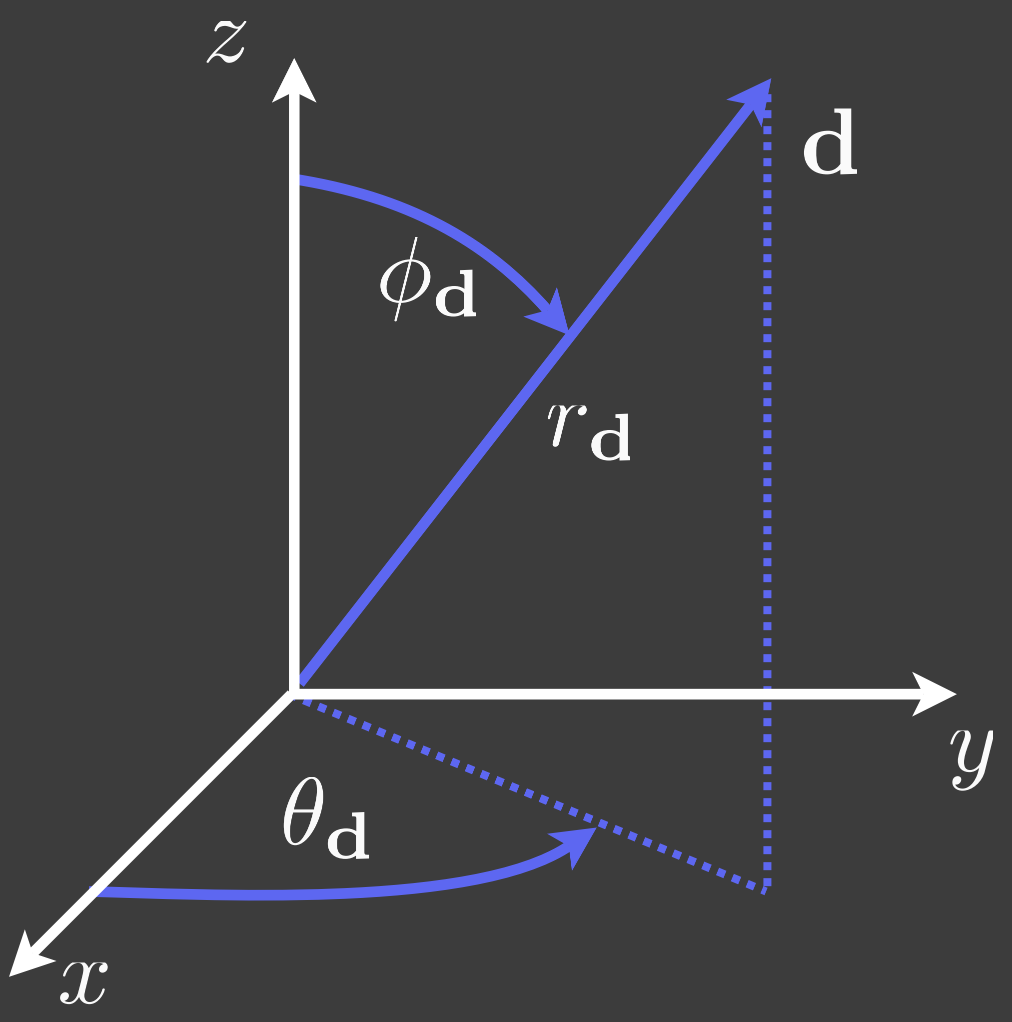
$$r = \sqrt{x^2 + y^2 + z^2}$$

$$y = \sin(\theta)\sin(\phi)$$

$$z = \cos(\phi)$$

$$\phi = \tan^{-1}\left(\frac{\sqrt{x^2 + y^2}}{z}\right)$$

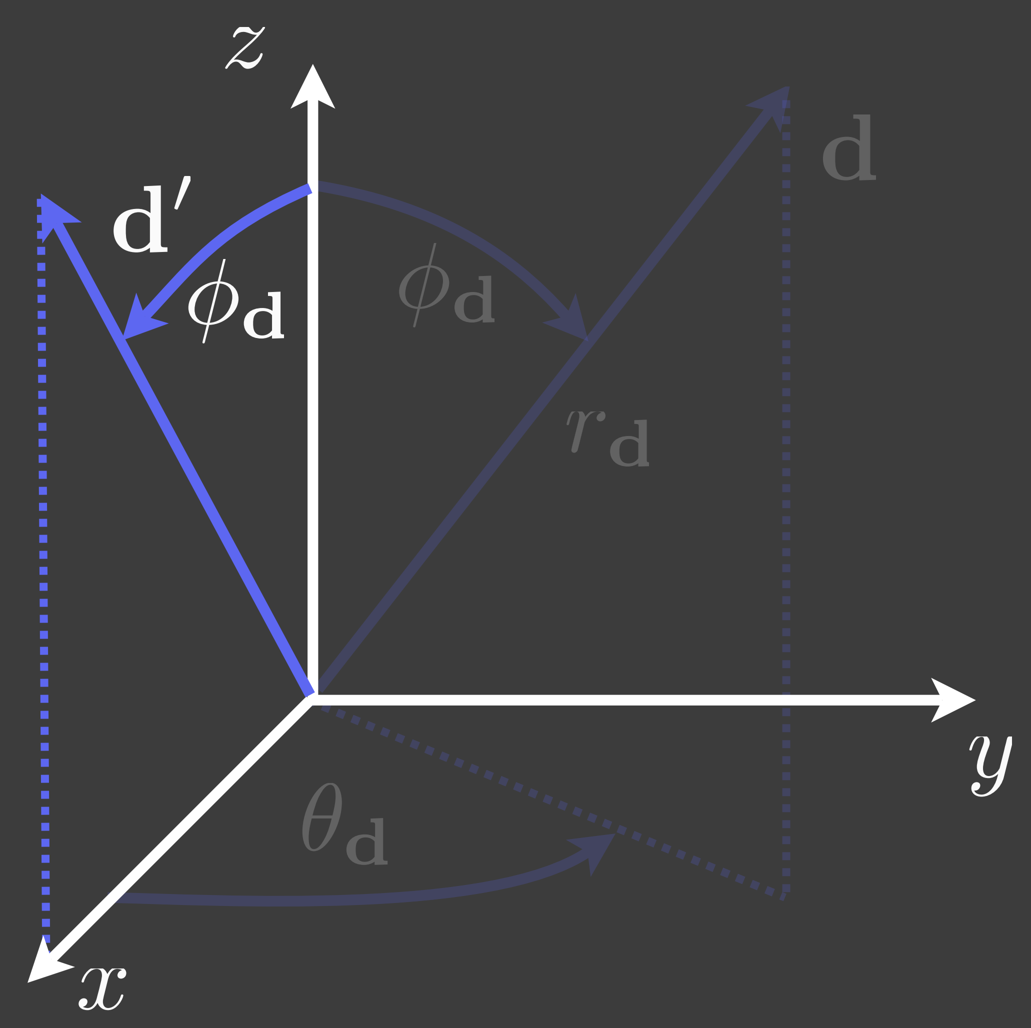
Use this to align the axis to z



Use this to align the axis to z

1) Bring $\mathbf{d}$ into the xz plane.

$$\mathbf{d}' = R_z(-\theta_{\mathbf{d}})\mathbf{d}$$



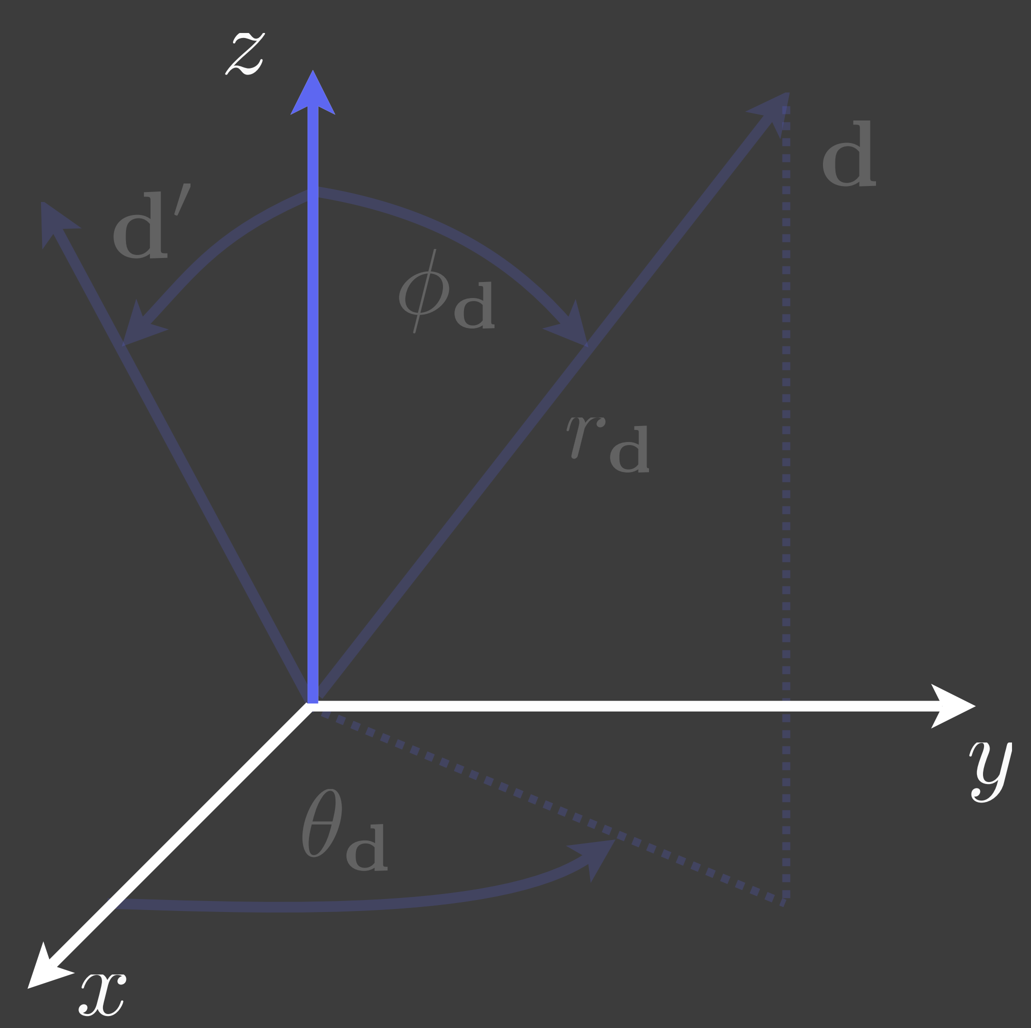
Use this to align the axis to z

1) Bring $\mathbf{d}$ into the xz plane.

$$\mathbf{d}' = R_z(-\theta_{\mathbf{d}})\mathbf{d}$$

2) Align $\mathbf{d}'$ to the z axis

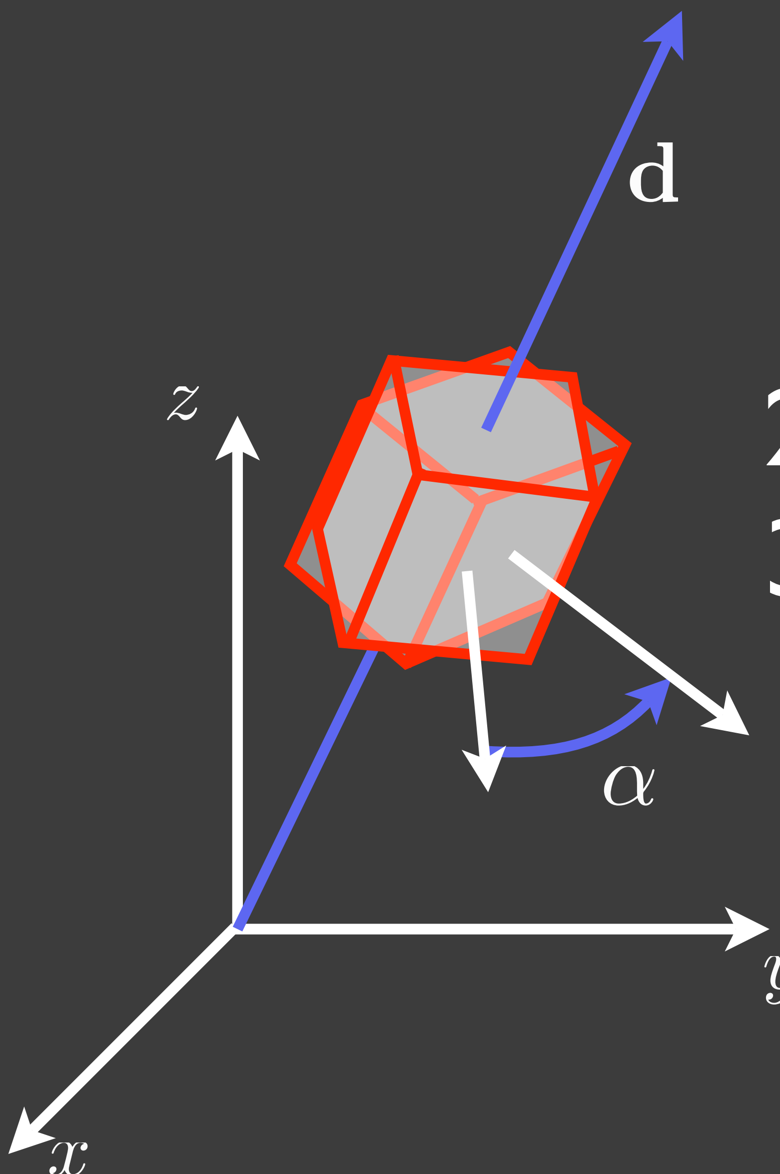
$$\begin{aligned}\mathbf{z} &= R_y(-\phi_{\mathbf{d}})\mathbf{d}' \\ &= R_y(-\phi_{\mathbf{d}})R_z(-\theta_{\mathbf{d}})\mathbf{d}\end{aligned}$$



The same transformation aligns all geometry:

$$\mathbf{p}' = R_y(-\phi_{\mathbf{d}})R_z(-\theta_{\mathbf{d}})\mathbf{p}$$

Back to Rotation About an Axis



1) Align d to z $\quad R_y(-\phi_{\mathbf{d}})R_z(-\theta_{\mathbf{d}})$

2) Rotate around it by α $\quad R_z(\alpha)$

3) Undo the Alignment

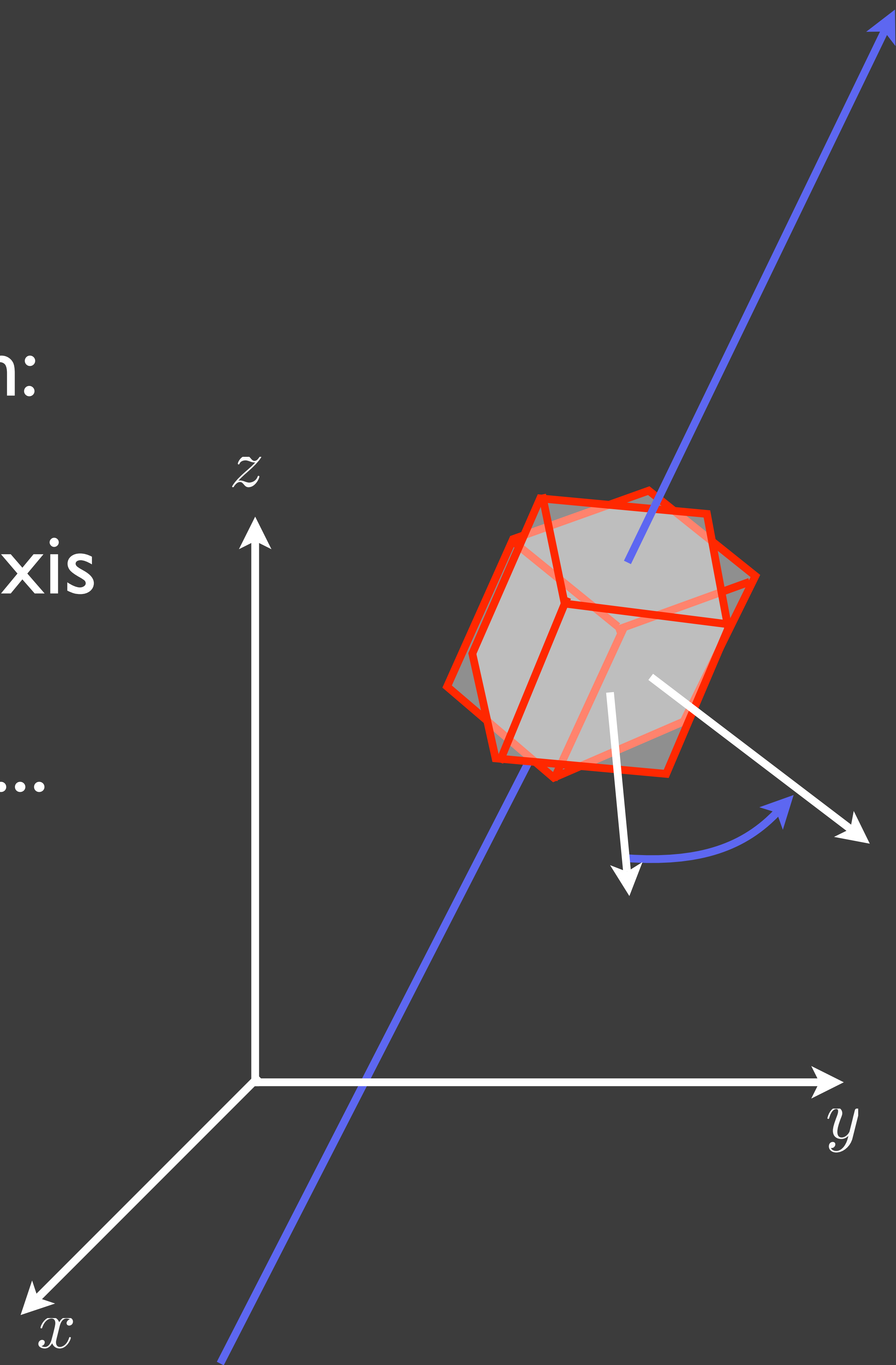
$$[R_y(-\phi_{\mathbf{d}})R_z(-\theta_{\mathbf{d}})]^{-1}$$
$$= R_z(-\theta_{\mathbf{d}})^{-1}R_y(-\phi_{\mathbf{d}})^{-1}$$
$$= R_z(\theta_{\mathbf{d}})R_y(\phi_{\mathbf{d}})$$

All Together:

$$\mathbf{p}' = \underbrace{R_z(\theta_{\mathbf{d}})R_y(\phi_{\mathbf{d}})R_z(\alpha)R_y(-\phi_{\mathbf{d}})R_z(-\theta_{\mathbf{d}})}_{R_{\mathbf{d}}(\alpha)}\mathbf{p}$$

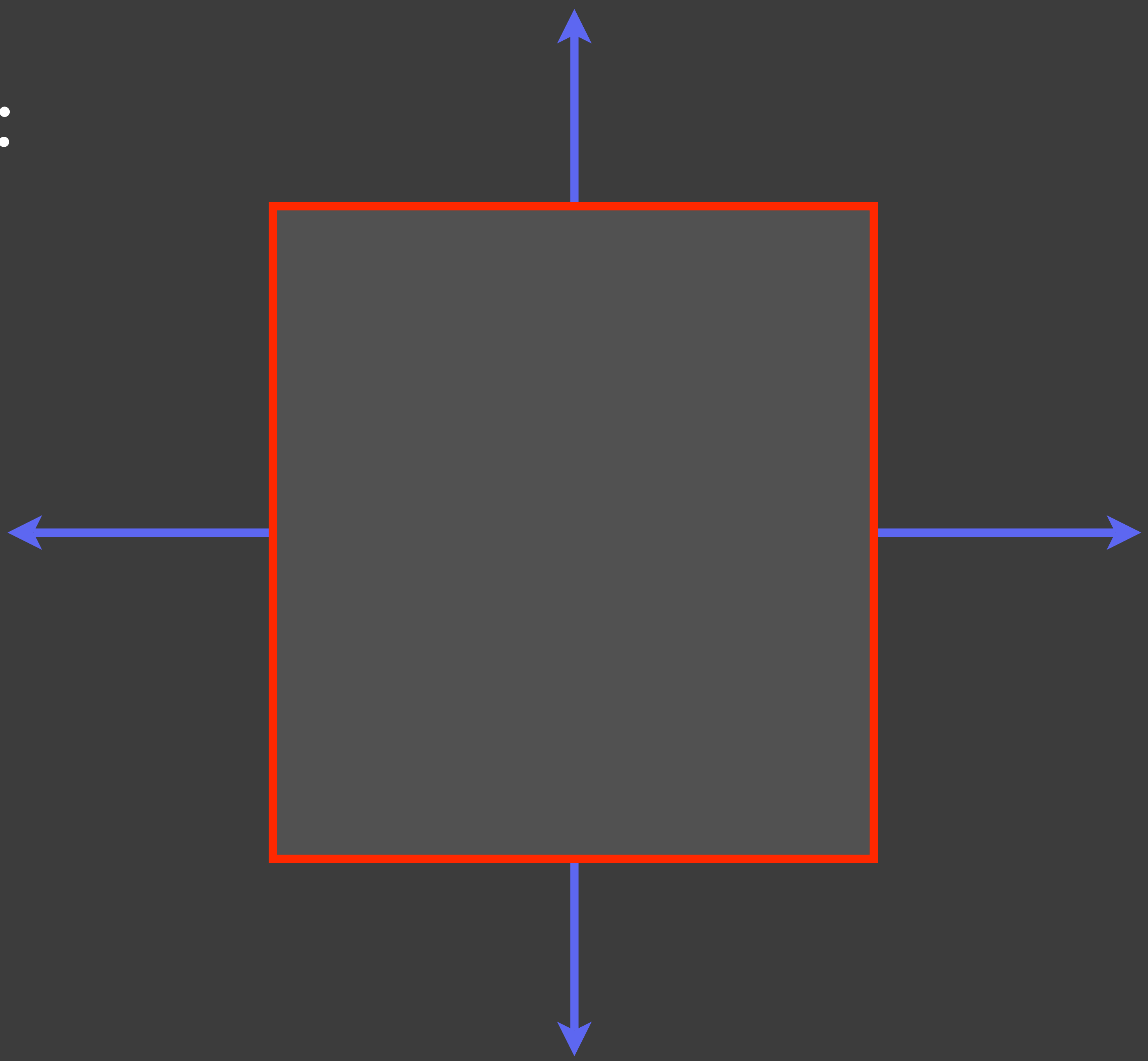
Thought Question:

Rotation about an axis that doesn't pass through the origin...



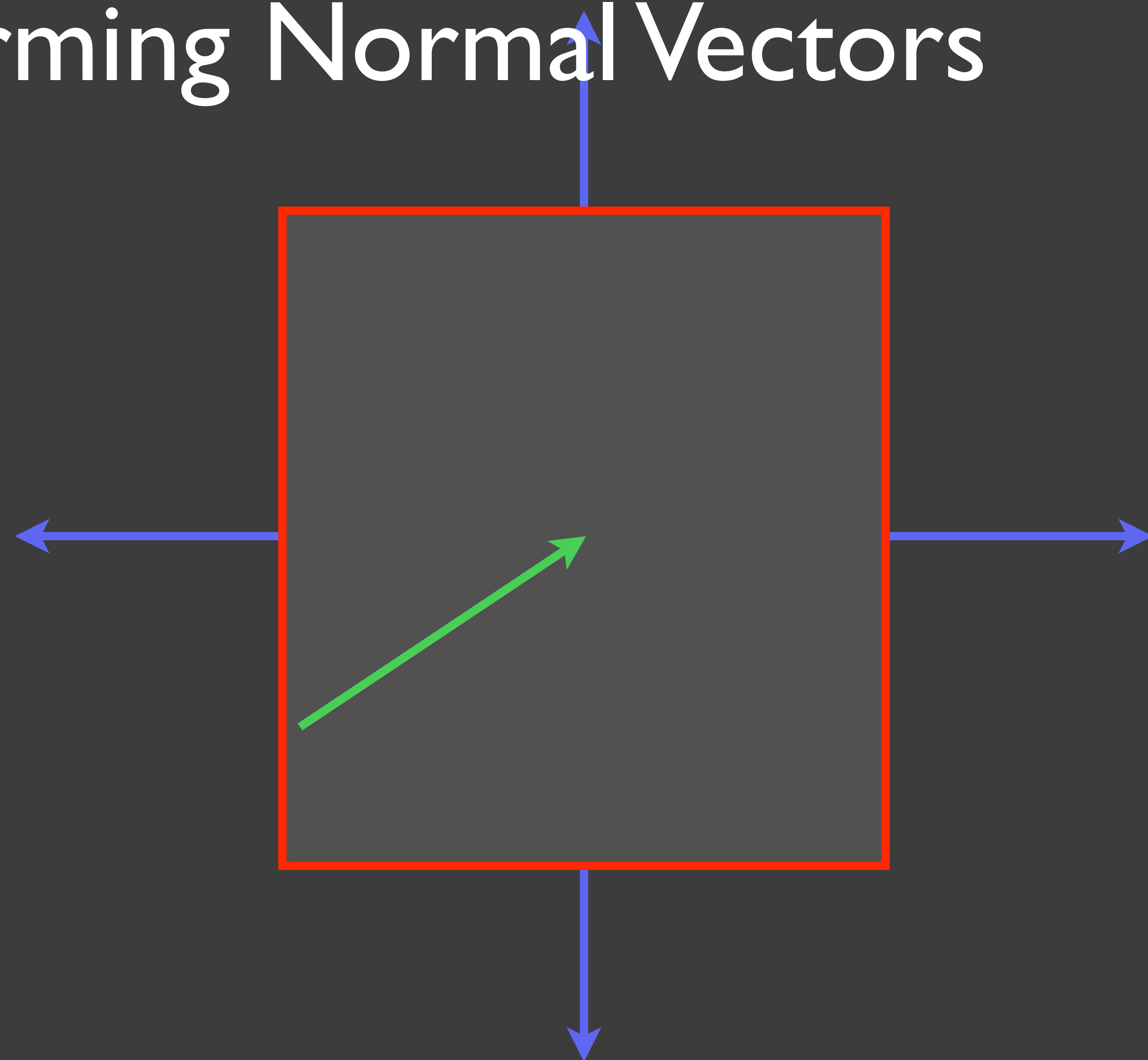
Transforming Normal Vectors

Translation:



Transforming Normal Vectors

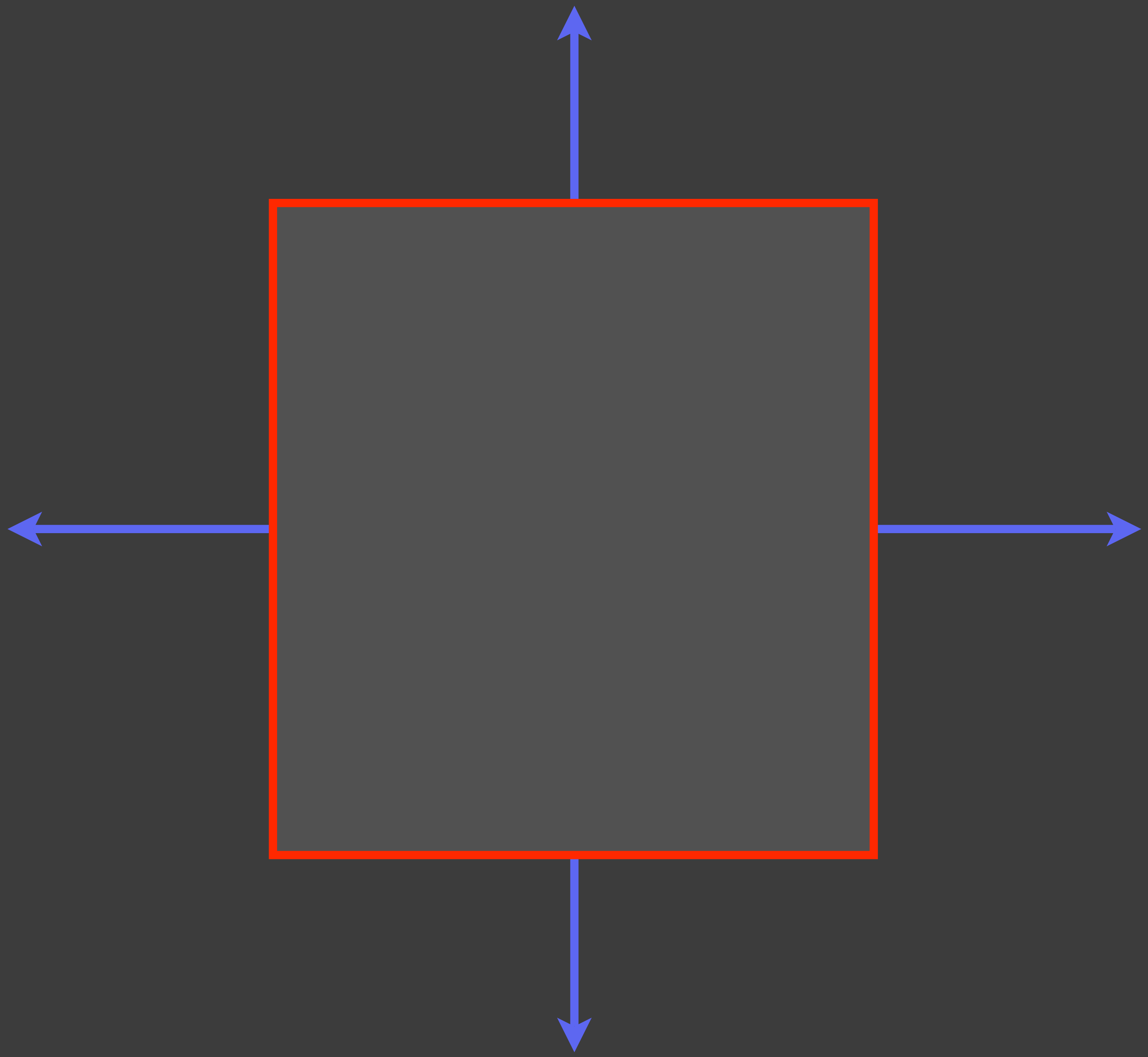
Translation:



(No effect)

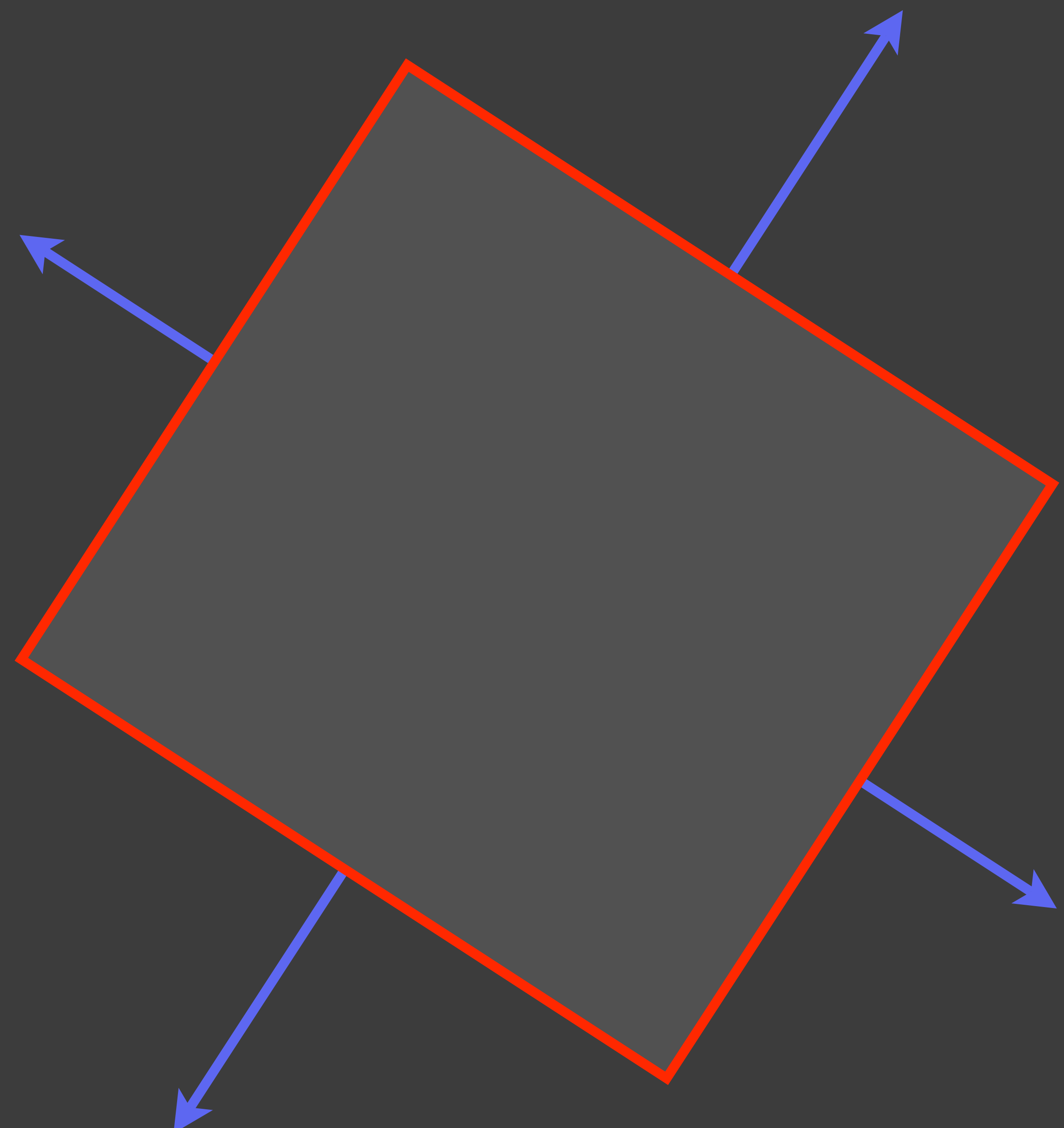
Transforming Normal Vectors

Rotation:



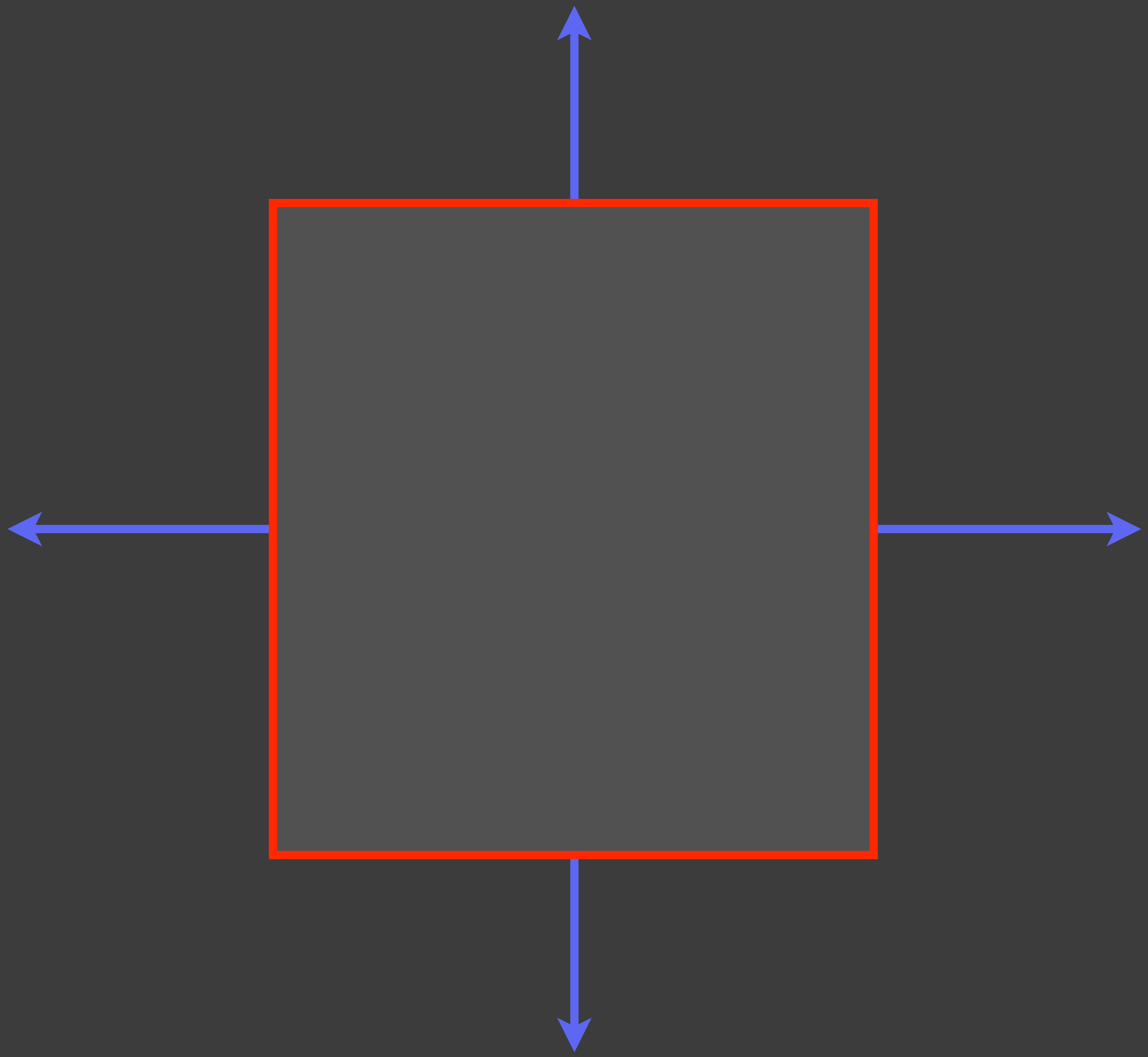
Transforming Normal Vectors

Rotation:



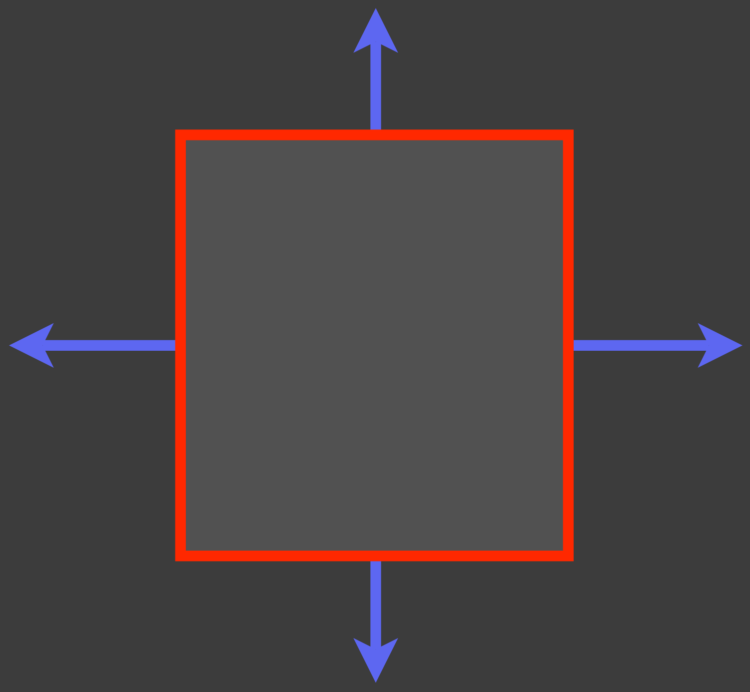
Transforming Normal Vectors

Scale:



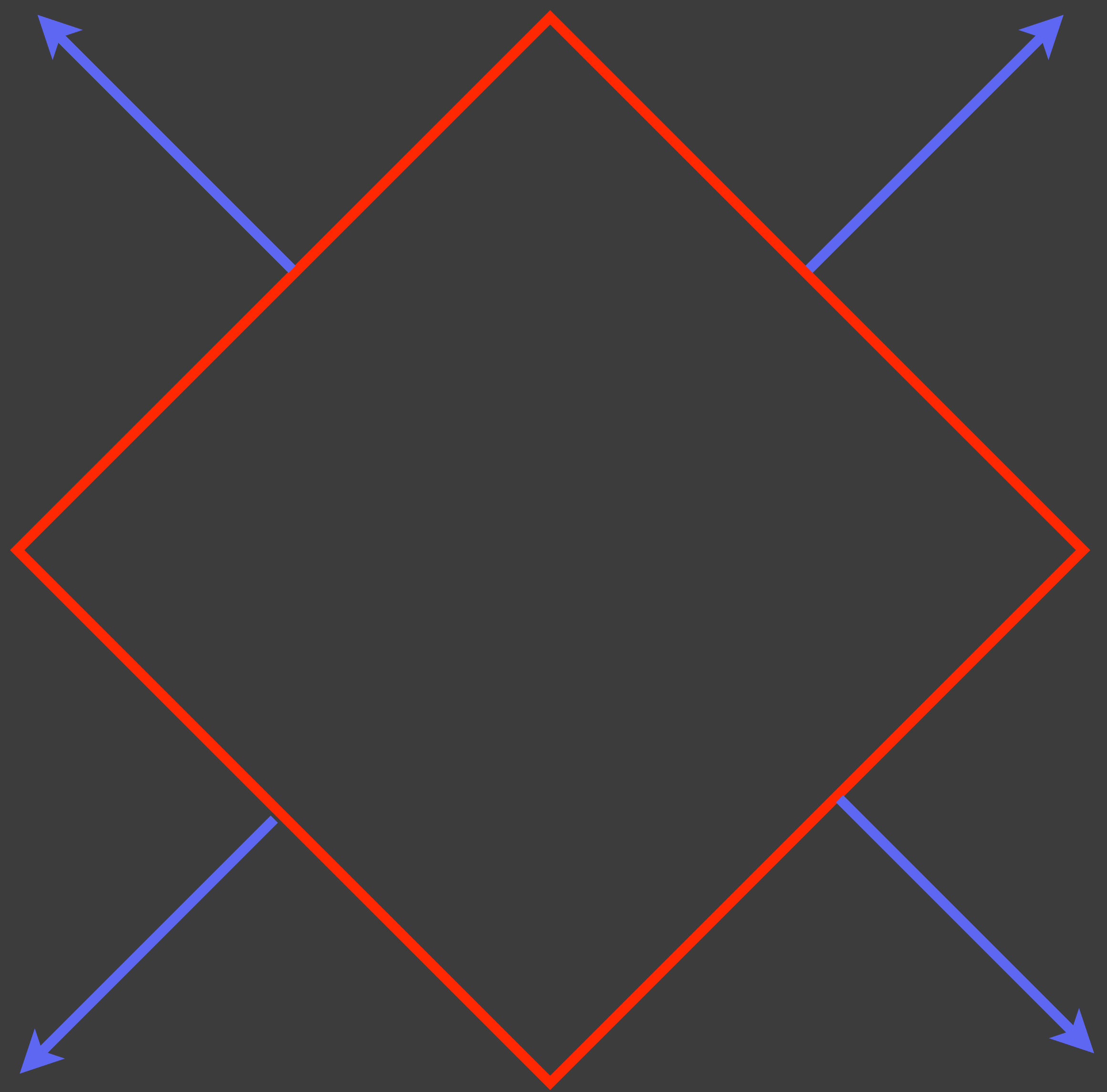
Transforming Normal Vectors

Scale:



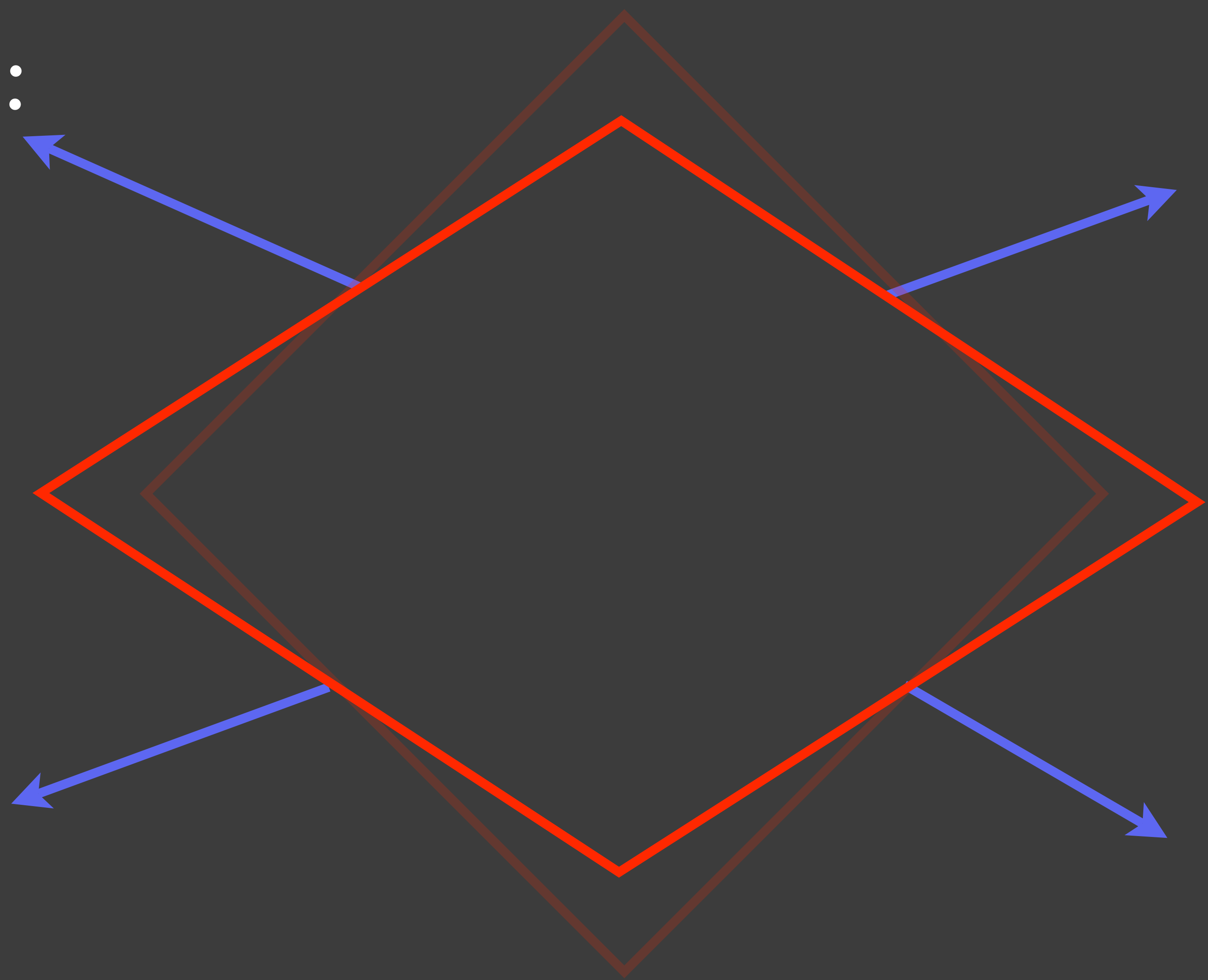
Transforming Normal Vectors

Scale, again:



Transforming Normal Vectors

Scale, again:

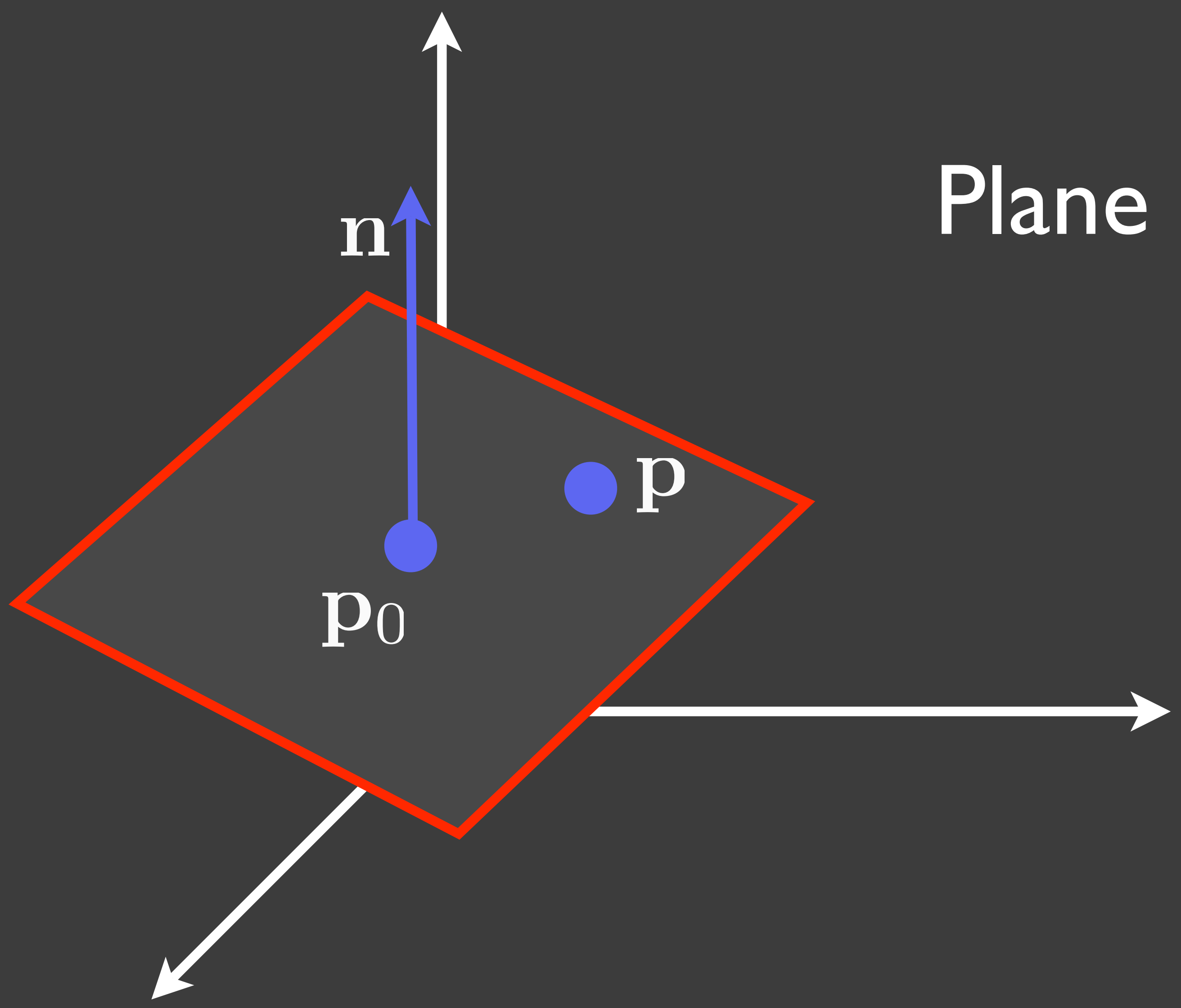


Transforming Normal Vectors

The line (2D) *defines* the normal vector.

We are really transforming the line (2D) or plane (3D)

Derivation



Plane (2D Line) Equation:

$$\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$$

or

$$\mathbf{n} \cdot \mathbf{a} = 0$$

Derivation

Plane

$$\mathbf{n} \cdot \mathbf{a} = 0$$

$$\mathbf{n}^T \mathbf{a} = 0$$

Transformed Plane

$$\mathbf{n}' \cdot \mathbf{a}' = 0$$

?

$$\mathbf{n}'^T \mathbf{a}' = 0$$

$$\mathbf{a}' = \mathbf{M}\mathbf{a} :$$

$$\mathbf{n}'^T \mathbf{M}\mathbf{a} = 0$$

$$\mathbf{n}^T \cancel{\mathbf{a}} = \mathbf{n}'^T \mathbf{M}\cancel{\mathbf{a}}$$

$$\mathbf{n}^T \mathbf{M}^{-1} = \mathbf{n}'^T$$

$$\mathbf{n}' = [\mathbf{n}^T \mathbf{M}^{-1}]^T = \boxed{{\mathbf{M}^{-1}}^T \mathbf{n}}$$

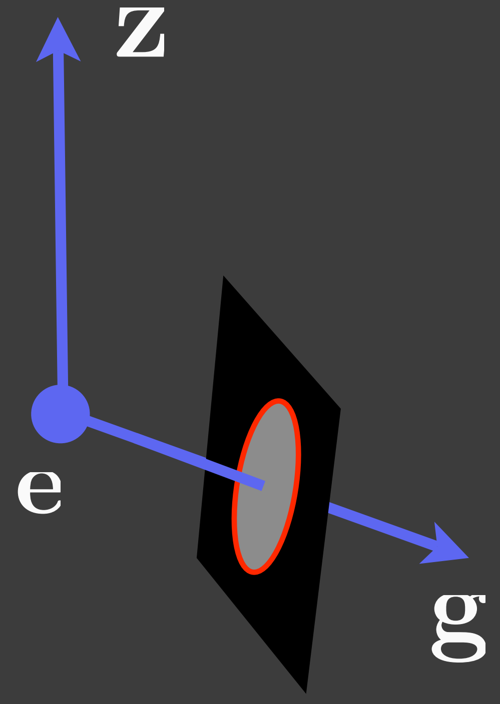
Rendering Overview

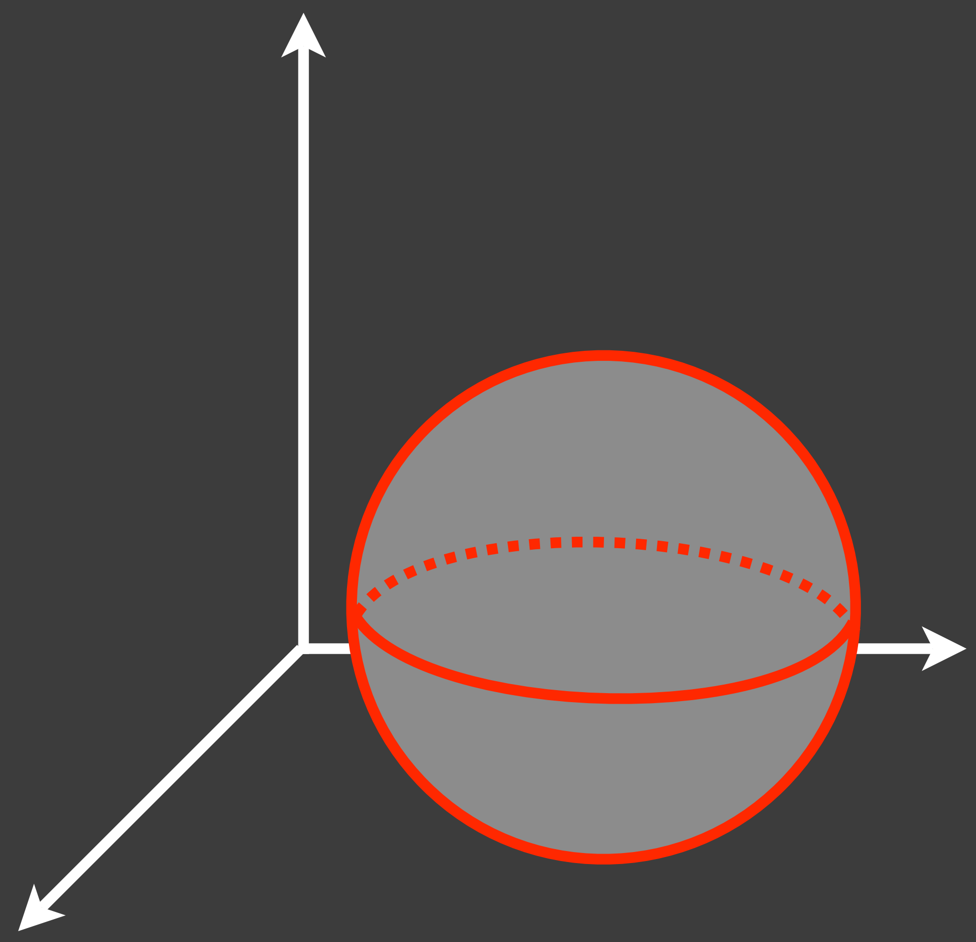
Camera:

eye $\mathbf{e}$

gaze $\mathbf{g}$

up $\mathbf{z}$





Lens?

Visibility?

Appearance?

Illumination?

Two Algorithmic Approaches

```
for each shape:
  for each pixel:
    if shape is visible:
      compute color
      store color in pixel
```

```
for each pixel:
  for each shape:
    if shape is visible:
      compute color
      store color in pixel
```

Two Algorithmic Approaches

```
for each shape:
  for each pixel:
    if shape is visible:
      compute color
      store color in pixel
```

Rasterization

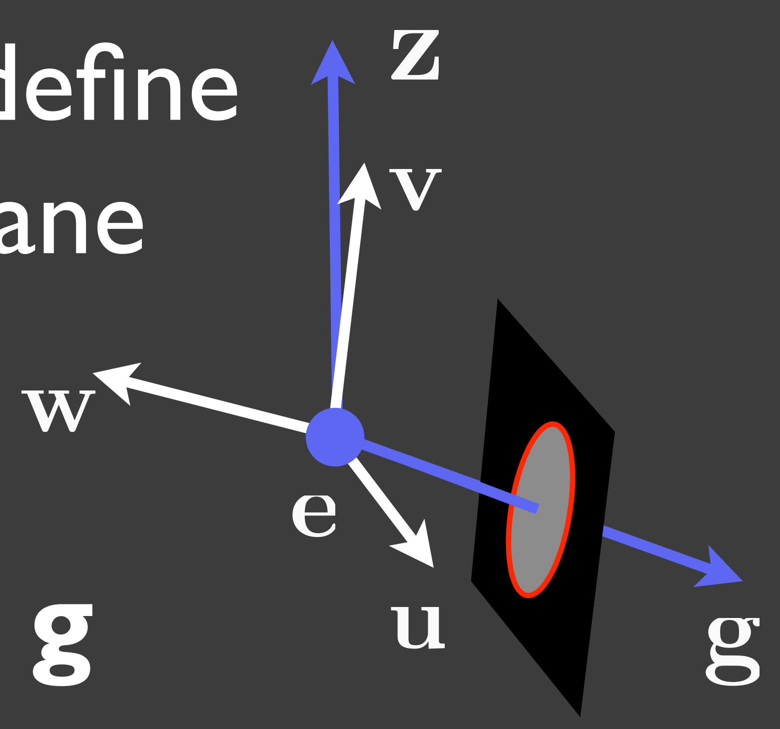
```
for each pixel:
  for each shape:
    if shape is visible:
      compute color
      store color in pixel
```

Ray Casting

The 3D Viewing Transformation

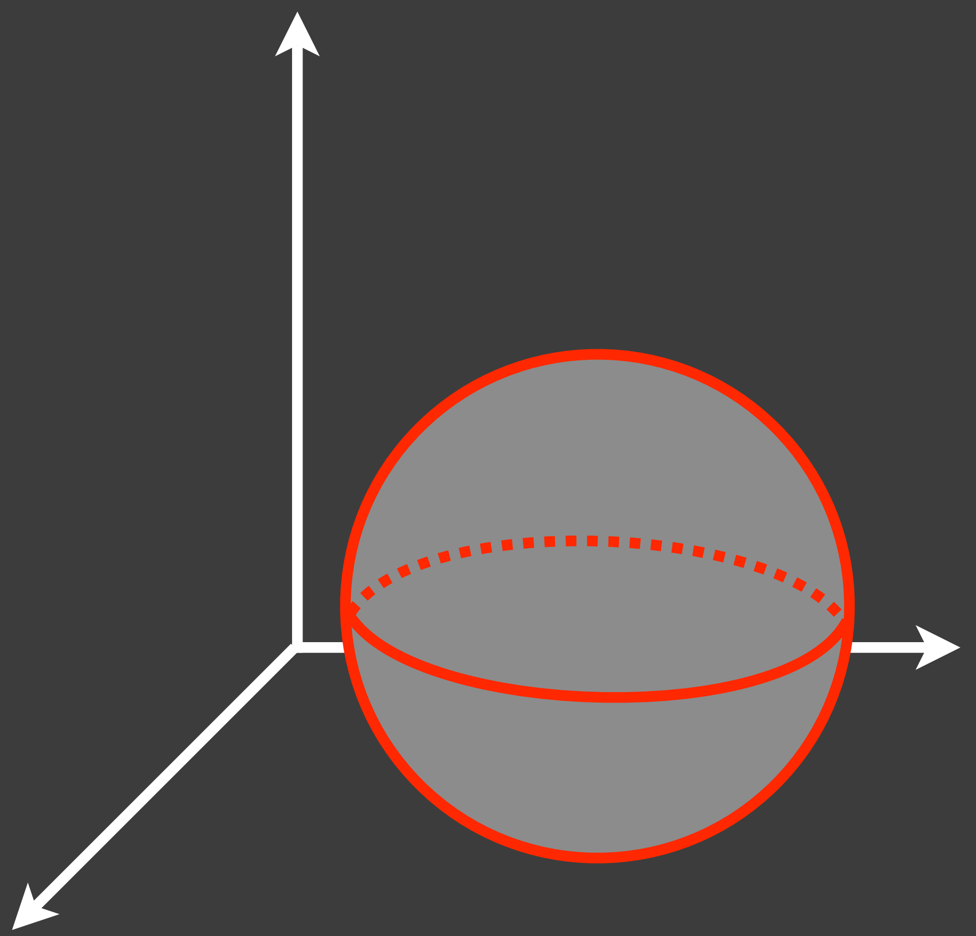
Goal: create a camera-centric coordinate system

u and **v** axes define the image plane



w opposes **g**

Orthonormal



The 3D Viewing Transformation

Constructing $(\mathbf{u}, \mathbf{v}, \mathbf{w})$

$\mathbf{w}$ opposes $\mathbf{g}$:

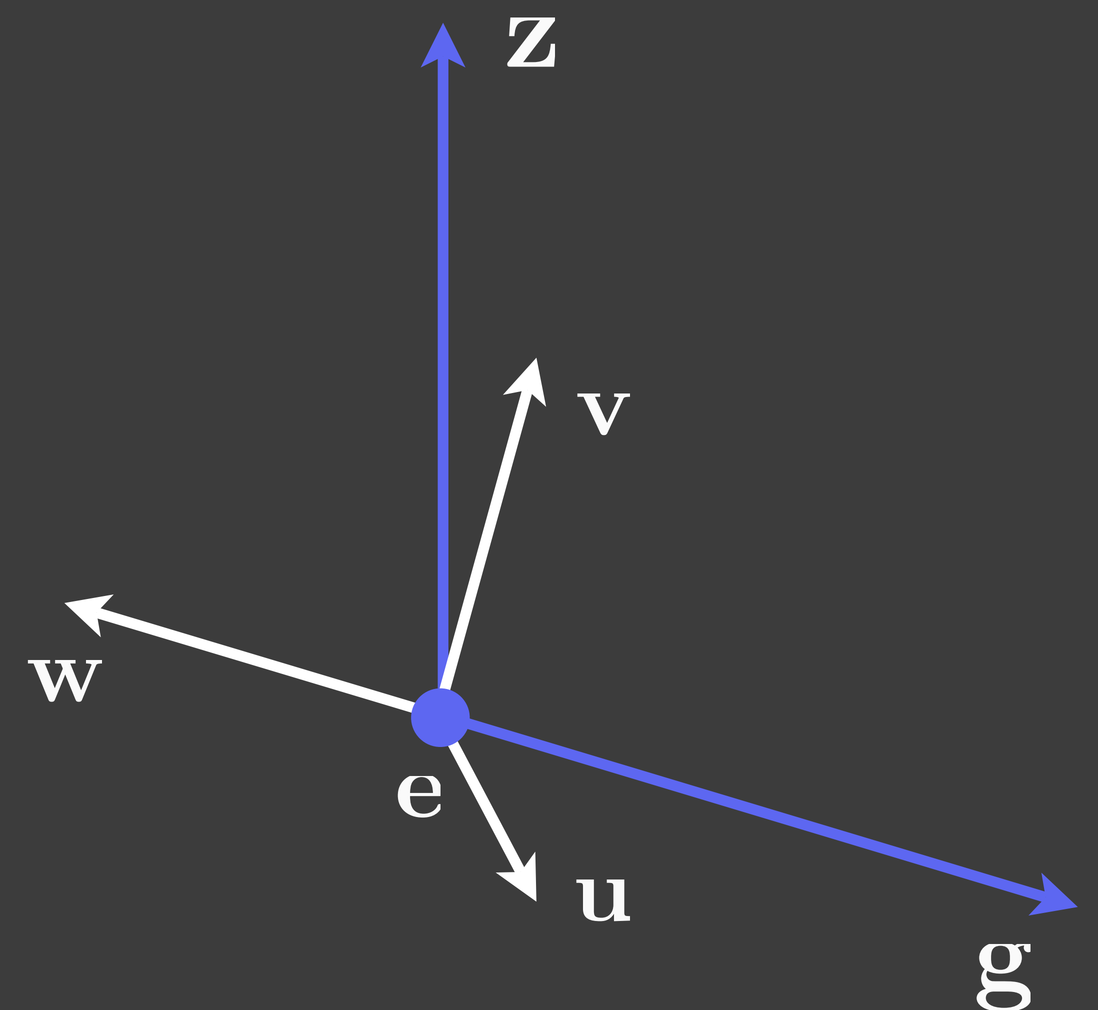
$$\mathbf{w} = -\frac{\mathbf{g}}{||\mathbf{g}||}$$

$\mathbf{u}$ is "right" when $\mathbf{z}$ is up:
$\mathbf{u}$ is perpendicular to $\mathbf{w}$:

$$\mathbf{u} = \frac{\mathbf{z} \times \mathbf{w}}{||\mathbf{z} \times \mathbf{w}||}$$

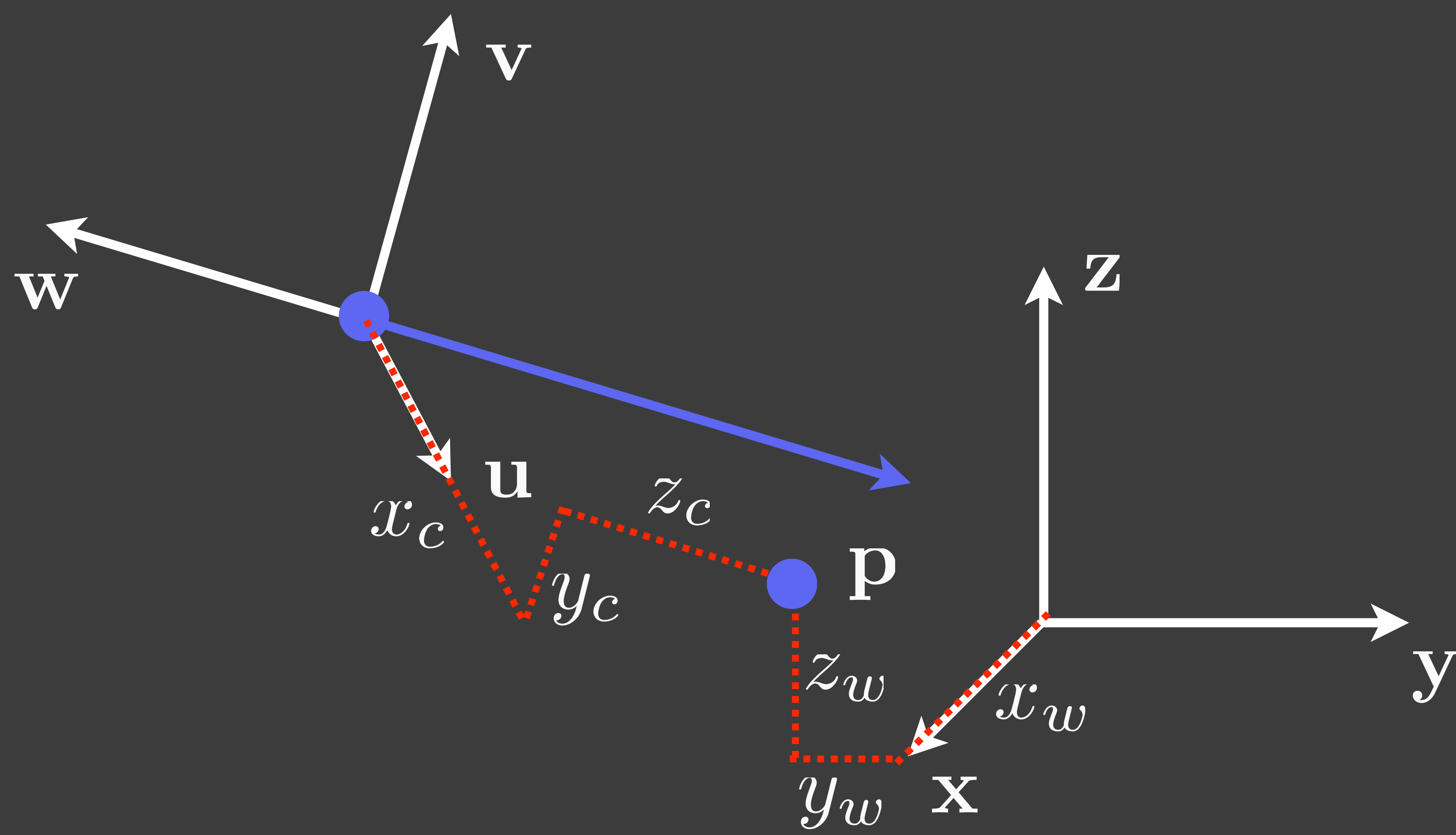
$\mathbf{v}$ is "up:"
$\mathbf{v}$ is perpendicular to $\mathbf{u}$ and $\mathbf{w}$:

$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$



Why is w backwards?

World Space and Camera Space



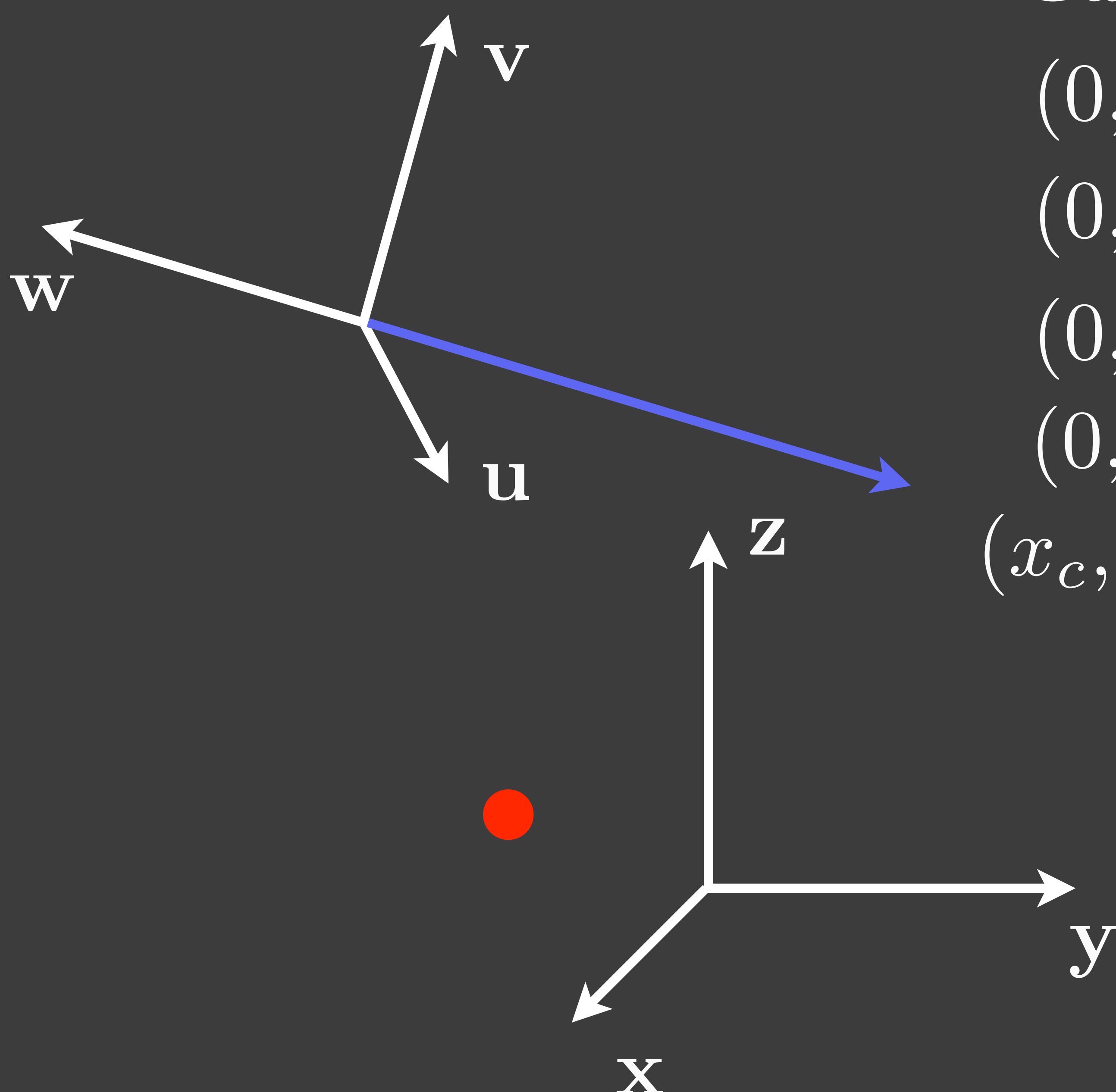
Camera space:

$$\mathbf{p}_c = (x_c, y_c, z_c)$$

World space:

$$\mathbf{p}_w = (x_w, y_w, z_w)$$

Camera Space to World Space



Camera	World
$(0,0,0)$	$\longrightarrow \mathbf{e}$
$(0,0,a)$	$\longrightarrow \mathbf{e} + a\mathbf{w}$
$(0,1,0)$	$\longrightarrow \mathbf{e} + \mathbf{v}$
$(0,1,a)$	$\longrightarrow \mathbf{e} + \mathbf{v} + a\mathbf{w}$
(x_c, y_c, z_c)	$\longrightarrow \mathbf{e} + x_c\mathbf{u} + y_c\mathbf{v} + z_c\mathbf{w}$

Camera Space to World Space

$$(x_c, y_c, z_c) \rightarrow \mathbf{e} + x_c\mathbf{u} + y_c\mathbf{v} + z_c\mathbf{w}$$

$$\mathbf{p}_w = \mathbf{e} + x_c\mathbf{u} + y_c\mathbf{v} + z_c\mathbf{w}$$

$$\mathbf{p}_w = \mathbf{e} + \begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix}\begin{bmatrix}x_c \\ y_c \\ z_c\end{bmatrix} = \mathbf{e} + \underbrace{\boxed{\begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix}}}_{\mathbf{A}_{cw}}\mathbf{p}_c$$

Camera Coordinates

$$\boxed{\mathbf{p}_w} = \mathbf{e} + \mathbf{A}_{cw}\boxed{\mathbf{p}_c}$$

World Coordinates

Camera Space to World Space

$$\mathbf{p}_w = \mathbf{e} + \mathbf{A}_{cw}\mathbf{p}_c \qquad \mathbf{A}_{cw} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

Homogeneous Transform:

$$\mathbf{p}_w = \underbrace{\begin{bmatrix} \begin{bmatrix} & \mathbf{A}_{cw} & \end{bmatrix} & \begin{bmatrix} \mathbf{e} \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix}}_{\mathbf{M}_{cw}} \mathbf{p}_c$$

World Space to Camera Space

Invert this: $\mathbf{p}_w = \mathbf{e} + \mathbf{A}_{cw}\mathbf{p}_c$

$$\mathbf{A}_{cw}\mathbf{p}_c = \mathbf{p}_w - \mathbf{e}$$

$$\mathbf{p}_c = \mathbf{A}_{cw}^{-1}(\mathbf{p}_w - \mathbf{e})$$

$$\mathbf{p}_c = \underbrace{\mathbf{A}_{cw}^{-1}}_{\text{Orthonormal}}\mathbf{p}_w - \underbrace{\mathbf{A}_{cw}^{-1}}_{\text{Orthonormal}}\mathbf{e}$$

$$\mathbf{p}_c = \mathbf{A}_{cw}^{T}\mathbf{p}_w - \mathbf{A}_{cw}^{T}\mathbf{e}$$

World Space to Camera Space

$$\mathbf{p}_c = \mathbf{A}_{cw}^T \mathbf{p}_w - \mathbf{A}_{cw}^T \mathbf{e}$$

Homogeneous:

$$\mathbf{p}_c = \begin{bmatrix} \begin{bmatrix} & \mathbf{A}_{cw}^T & \\ & & \end{bmatrix} & \begin{bmatrix} -\mathbf{A}_{cw}^T \mathbf{e} \\ \end{bmatrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \mathbf{p}_w$$

$\mathbf{M}_{wc}$, the viewing transformation

$$\mathbf{M}_{wc} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Camera Space to World Space, Again

Translation

$$\mathbf{p}_w = \boxed{\mathbf{e}} + \boxed{\mathbf{A}_{cw}}\mathbf{p}_c$$

Orthonormal

3x3 Orthonormal matrices *are* 3D rotation matrices

You can specify a camera using translation and rotation of proxy geometry that looks like a camera, much as you would manipulate a real camera, and build the same viewing transform!