

CSCI 420 Computer Graphics

Lecture 5

# Transformations

Vector Spaces  
Euclidean Spaces  
Frames  
Homogeneous Coordinates  
Transformation Matrices  
[Angel, Ch. 3]

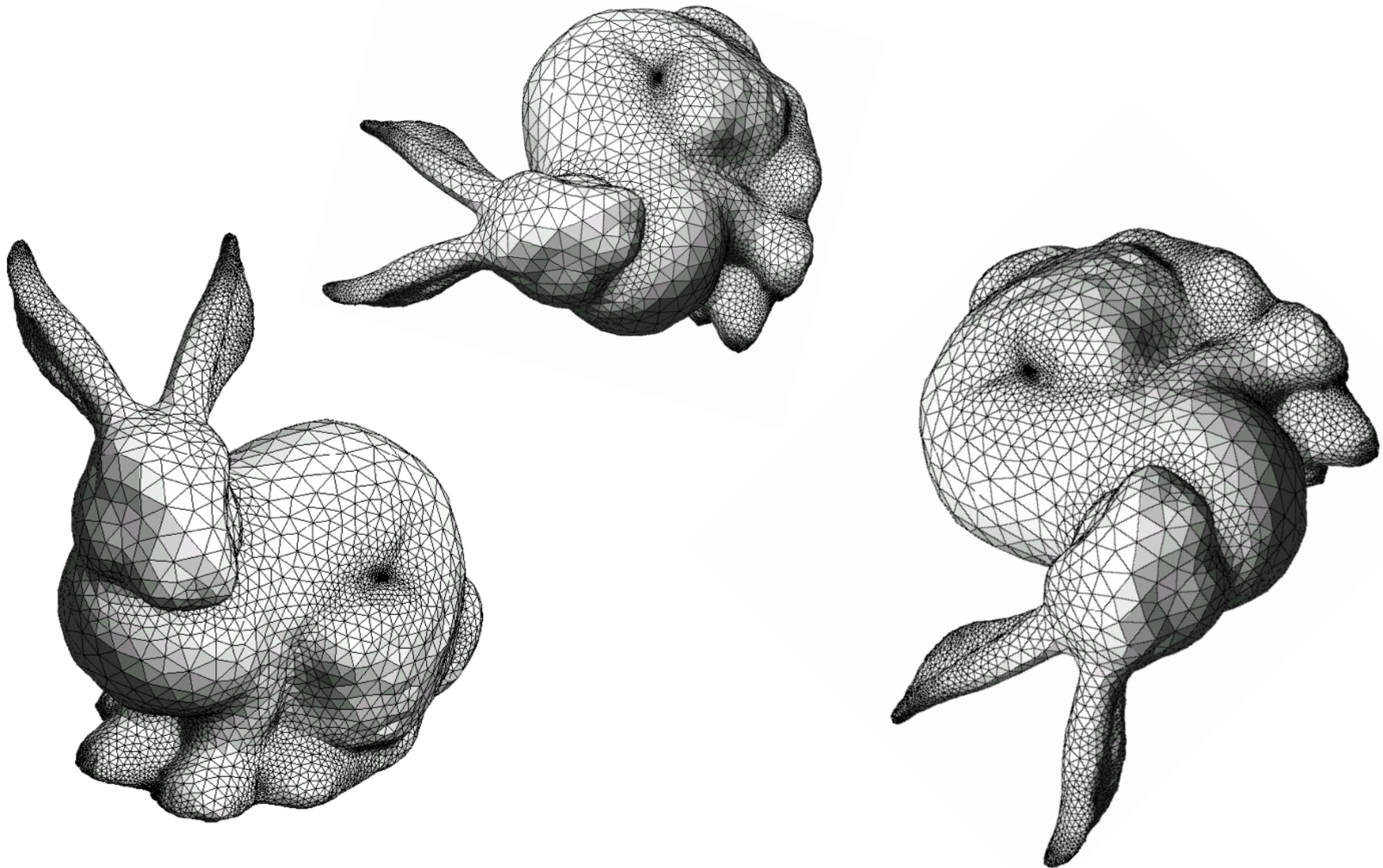
Oded Stein

University of Southern California

# A Preview of an OpenGL Application

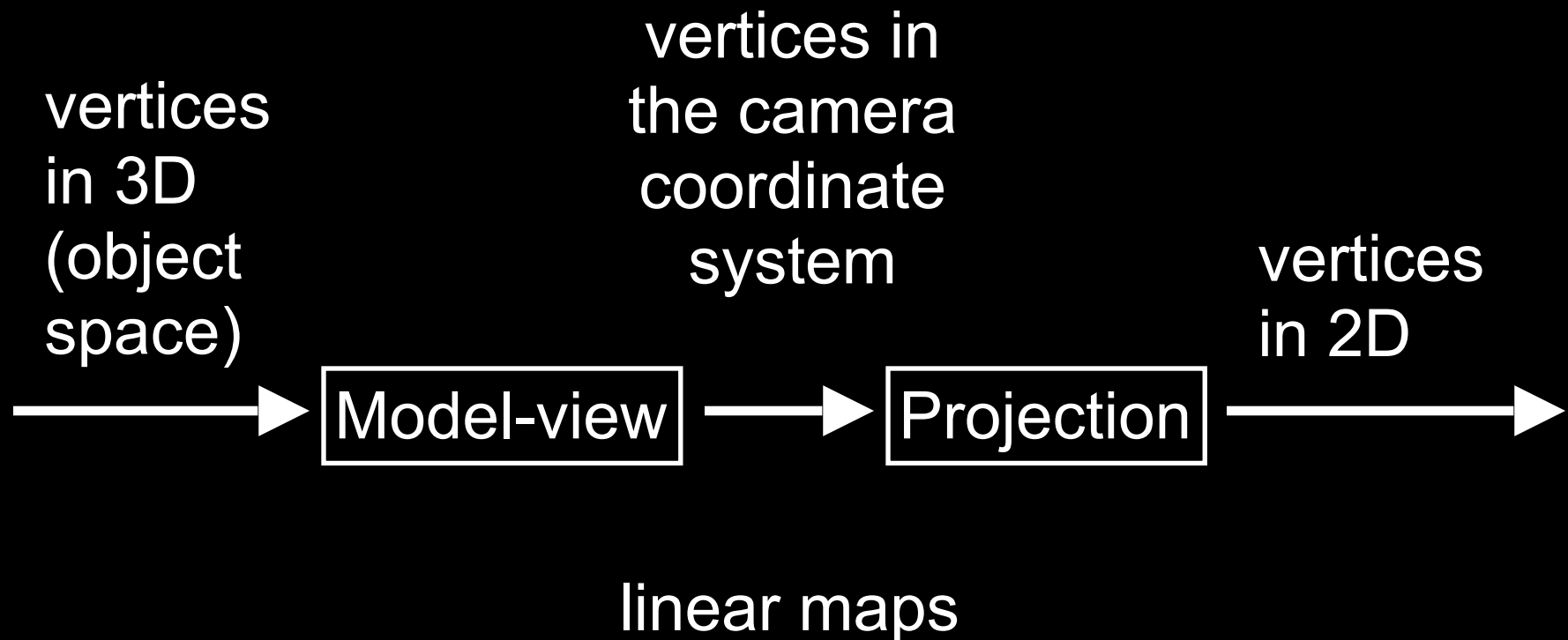
- Some parts will not make sense yet
  - In the future: shaders
- HW1 code is still being updated
- Might help you have something more concrete to look at!
- Will be uploaded to course website.

# OpenGL Transformations



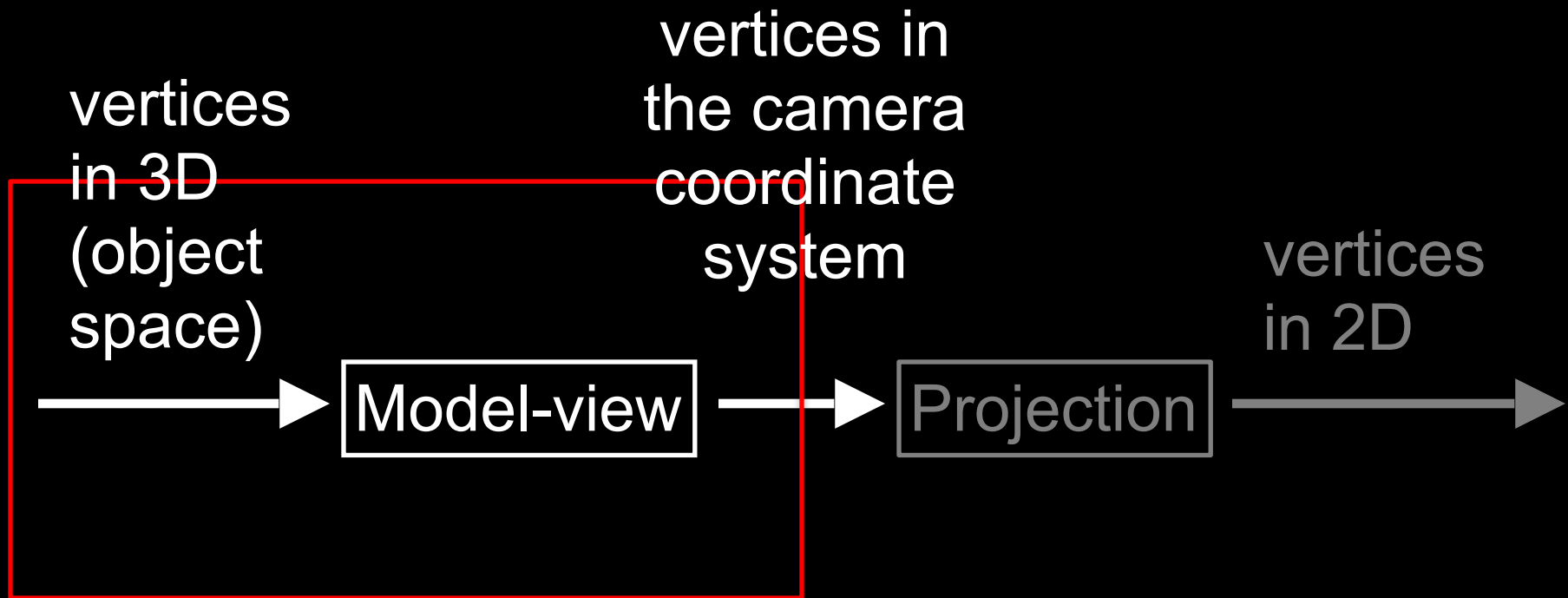
# OpenGL Transformation Matrices

- Model-view matrix (4x4 matrix)
- Projection matrix (4x4 matrix)



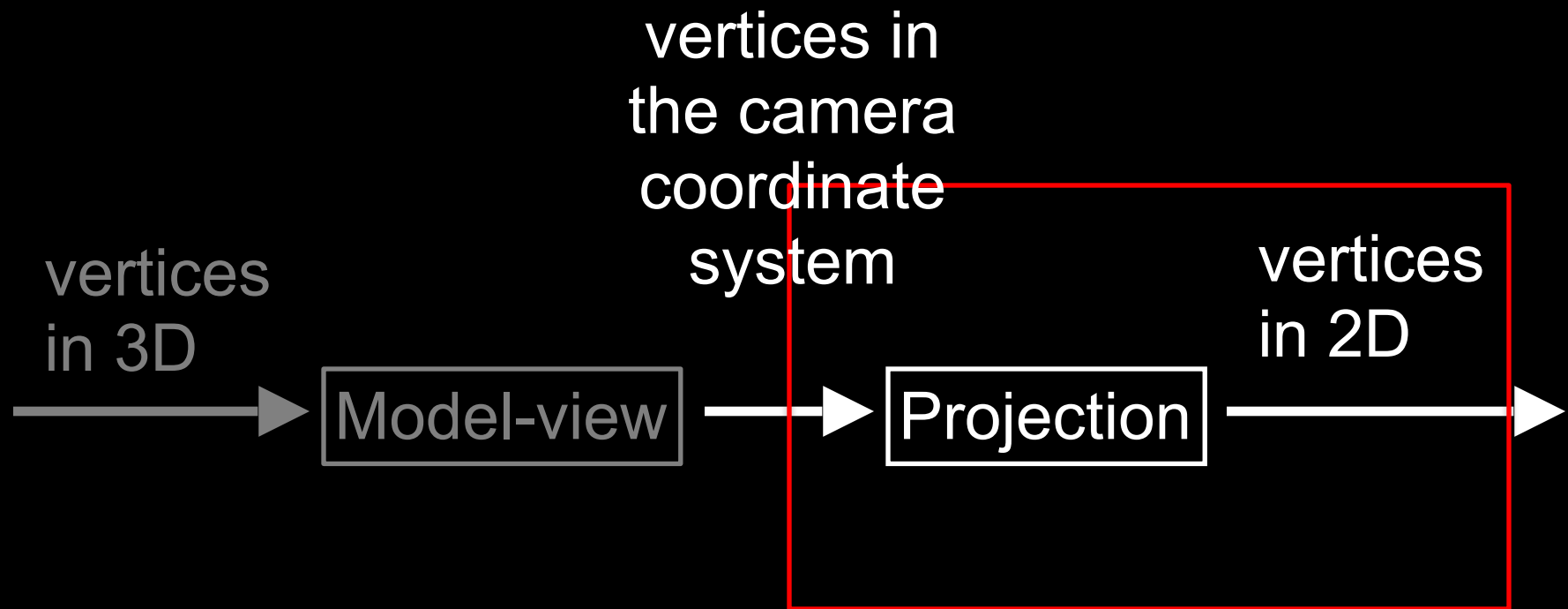
# 4x4 Model-view Matrix (this lecture)

- Translate, rotate, scale objects
- Position the camera



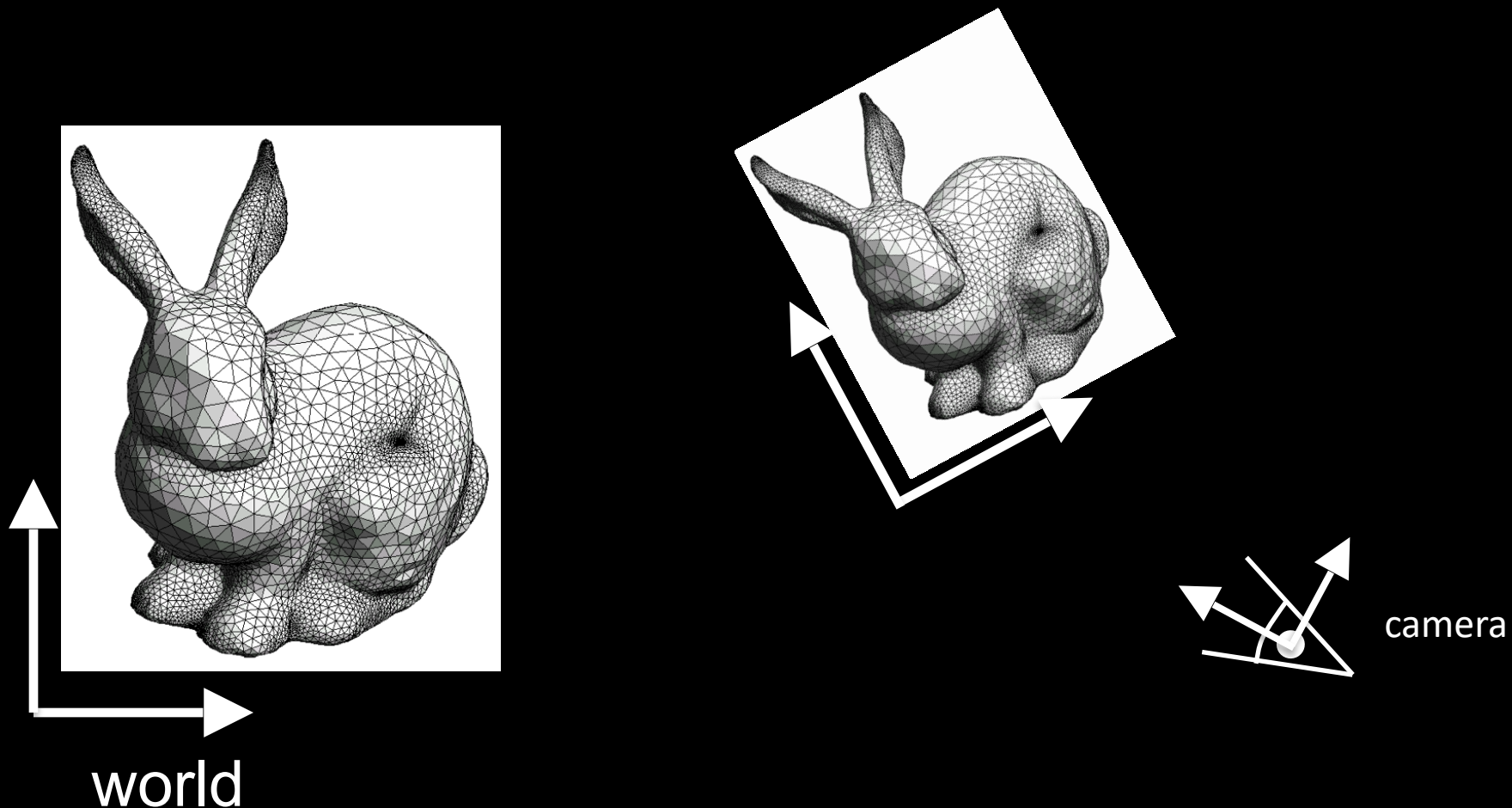
# 4x4 Projection Matrix (next lecture)

- Project from 3D to 2D



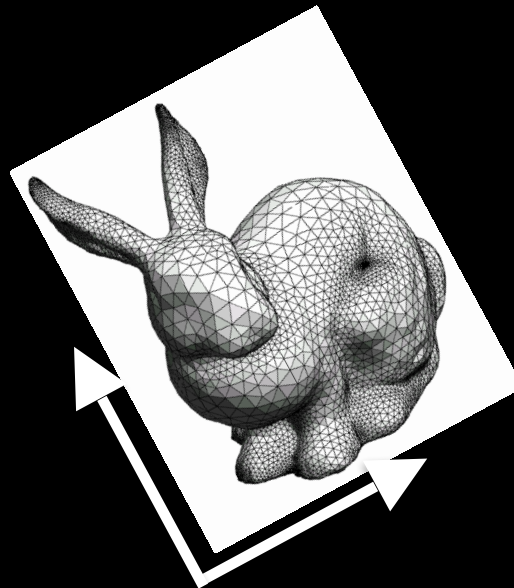
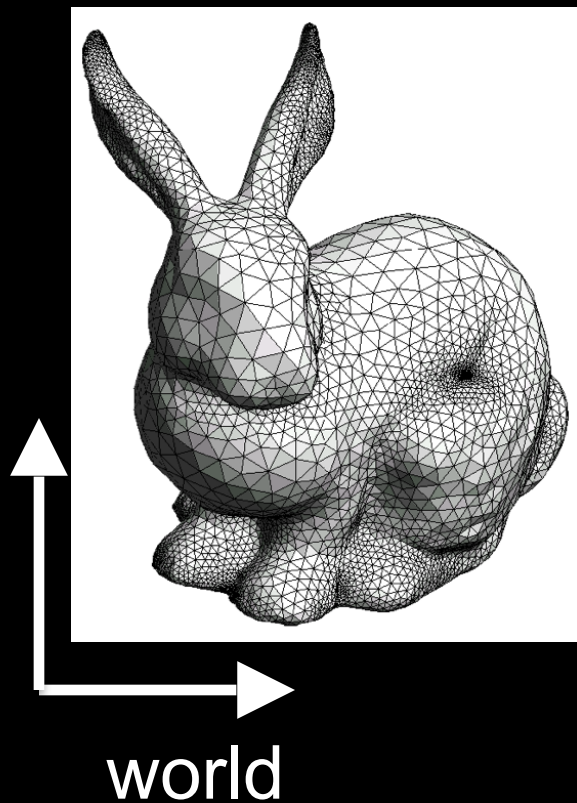
# 4x4 Model-view Matrix (this lecture)

- Translate, rotate, scale objects in world space
- Position and orient the camera



# 4x4 Model Matrix

- Translate, rotate, scale objects in world space





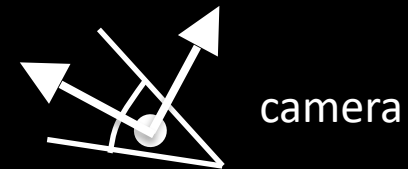
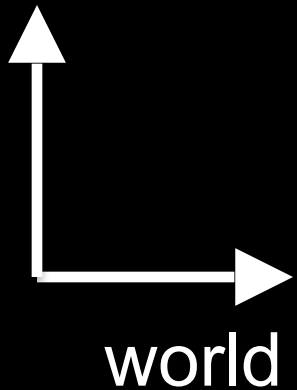
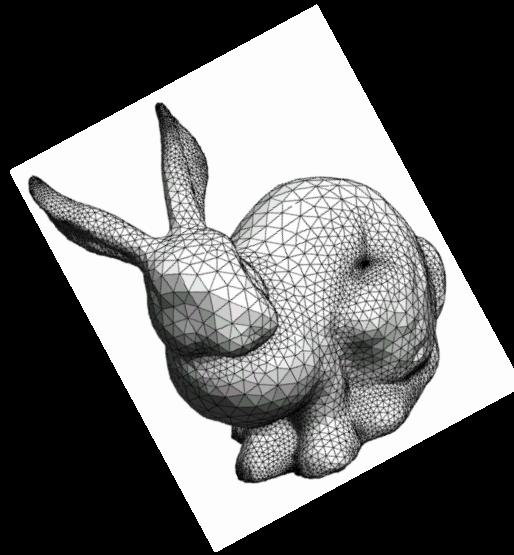
# 4x4 Model Matrix

- Translate, rotate, scale objects in world space
- The user rotated the object, but we don't want to reload the object.



# 4x4 View Matrix

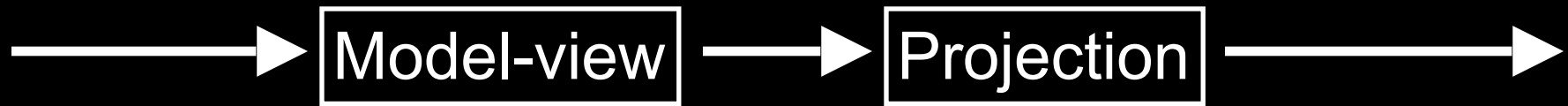
- Position and orient the camera
- From world space to camera space



# 4x4 Model-view Matrix

- These two do more or less the same thing.
- Rigid maps in 3D that position things.
- Difference between "model" (moving around objects) and "view" (moving around the camera) is purely a convention.
- In the OpenGL model, they are often not distinguished at all.

# OpenGL Transformation Matrices



- Manipulated separately in OpenGL
- Core profile: set them directly
  - You will load the matrices into memory, and apply them yourself in a shader.
- Compatibility profile: must set matrix mode

```
glMatrixMode (GL_MODELVIEW);  
glMatrixMode (GL_PROJECTION);
```

# Setting the Modelview Matrix: Core Profile

- Set identity:

```
openGLMatrix.SetMatrixMode (OpenGLMatrix::ModelView);  
openGLMatrix.LoadIdentity();
```

- Use our openGLMatrix library functions:

```
openGLMatrix.Translate(dx, dy, dz);  
openGLMatrix.Rotate(angle, vx, vy, vz);  
openGLMatrix.Scale(sx, sy, sz);
```

- Upload m to the GPU:

```
float m[16]; // column-major  
openGLMatrix.GetMatrix(m);  
GLboolean isRowMajor = GL_FALSE;  
pipelineProgram->SetUniformVariableMatrix4fv(  
    "modelViewMatrix", isRowMajor, m);
```

# Setting the Modelview Matrix: Core Profile

- Set identity:

```
openGLMatrix.SetMatrixMode (OpenGLMatrix::ModelView);  
openGLMatrix.LoadIdentity();
```

- Use our openGLMatrix library functions:

```
openGLMatrix.Translate(dx, dy, dz);  
openGLMatrix.Rotate(angle, vx, vy, vz);  
openGLMatrix.Scale(sx, sy, sz);
```

- Upload m to the GPU:

```
float m[16]; // column-major  
openGLMatrix.GetMatrix(m);  
GLboolean isRowMajor = GL_FALSE;  
pipelineProgram->SetUniformVariableMatrix4fv(  
    "modelViewMatrix", isRowMajor, m);
```

what is this?

# Setting the Modelview Matrix: Core Profile

- **ALTERNATIVELY:** do your own linear algebra!

```
float m[16]; // column-major
m[0] = ...
...
m[15] = ...
GLboolean isRowMajor = GL_FALSE;
pipelineProgram->SetUniformVariableMatrix4fv(
    "modelViewMatrix", isRowMajor, m);
```

- You can multiply with any matrix here.
  - Do more than just boring old rigid maps.

# Setting the Modelview Matrix: Compatibility Profile

- Load or post-multiply

```
glMatrixMode (GL_MODELVIEW);  
glLoadIdentity(); // very common usage  
float m[16] = { ... };  
glLoadMatrixf(m); // rare, advanced  
glMultMatrixf(m); // rare, advanced
```

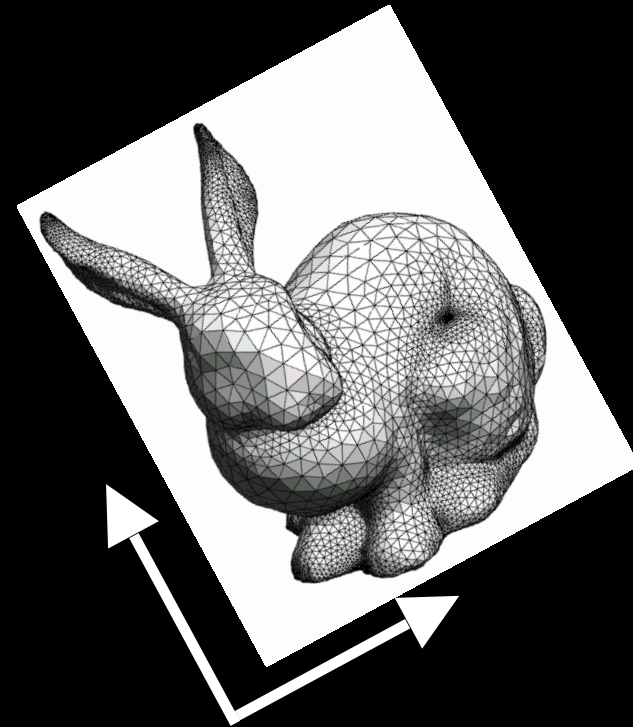
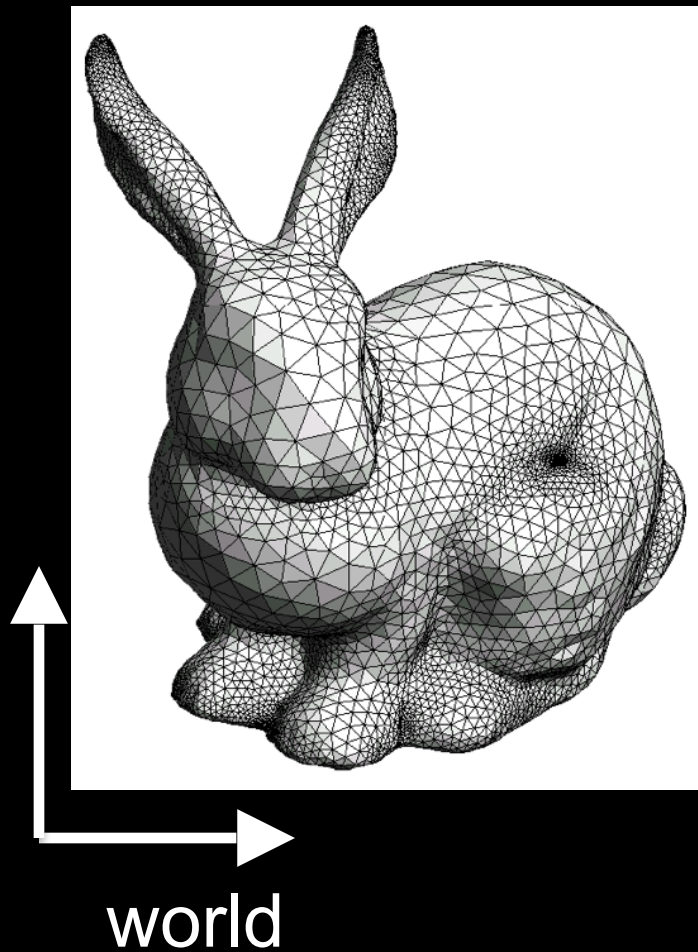
- Use library functions

```
glTranslatef(dx, dy, dz);  
glRotatef(angle, vx, vy, vz);  
glScalef(sx, sy, sz);
```

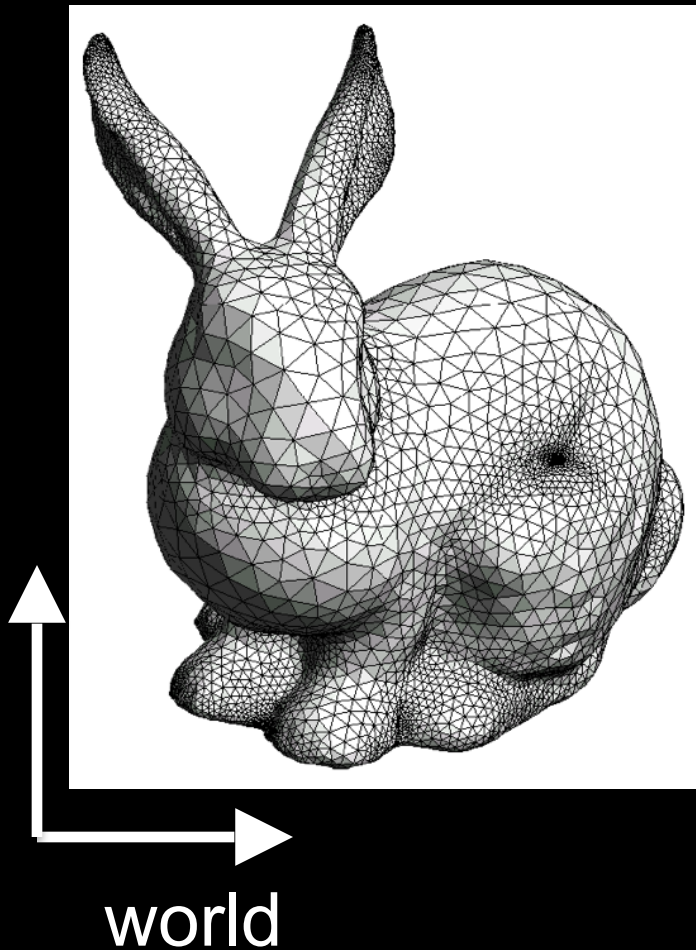
- More intuitive at first, but causes headaches later.
- No control over matrix.



# Translated, rotated, scaled object



# The *rendering* coordinate system

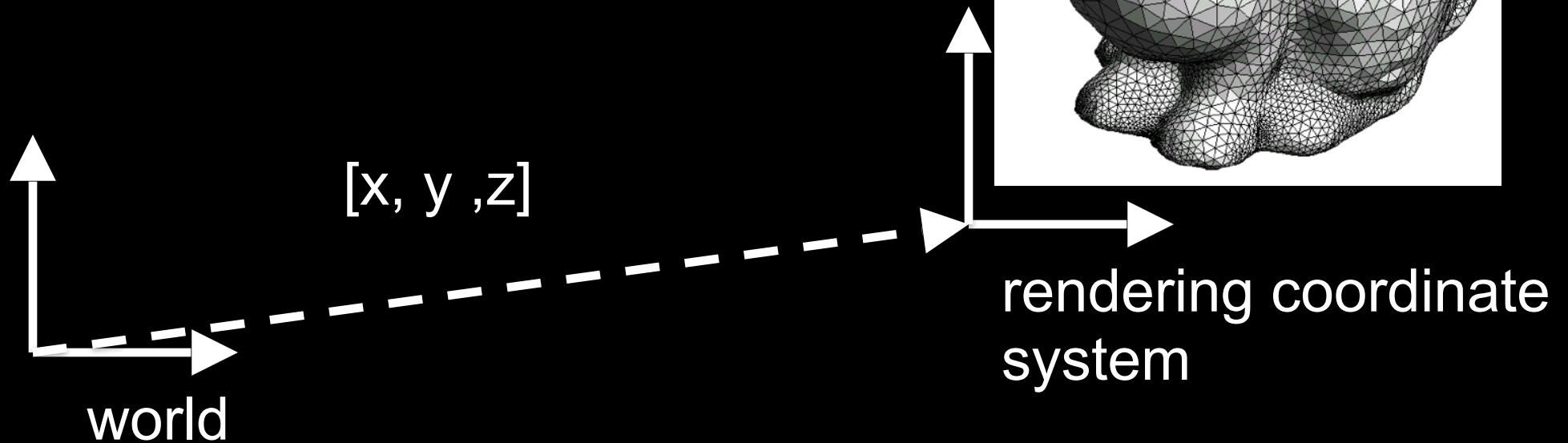


Initially (after `LoadIdentity()`) :

rendering coordinate system =  
world coordinate system

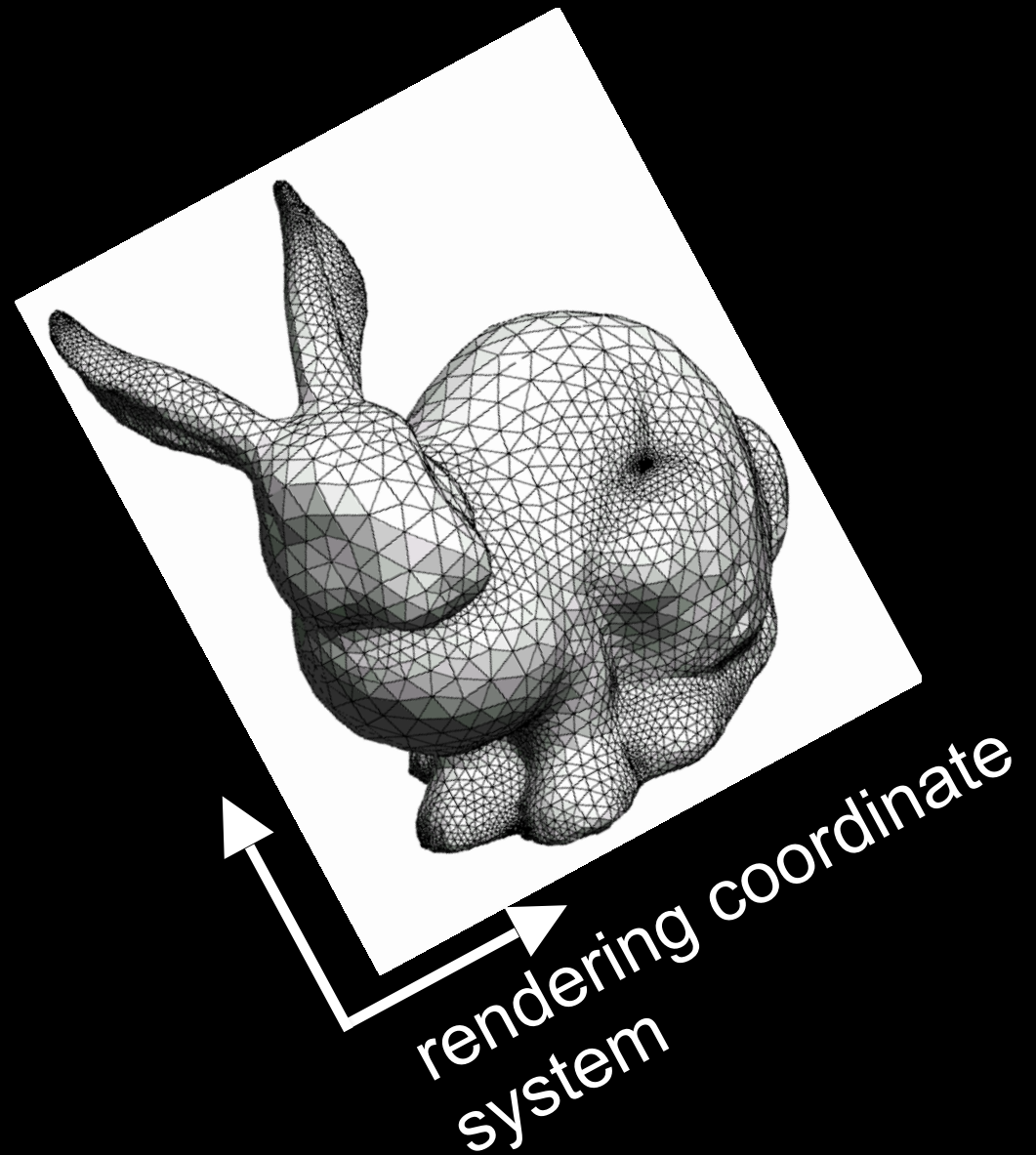
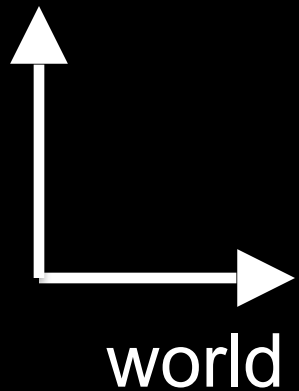
# The *rendering* coordinate system

```
Translate(x, y, z);
```



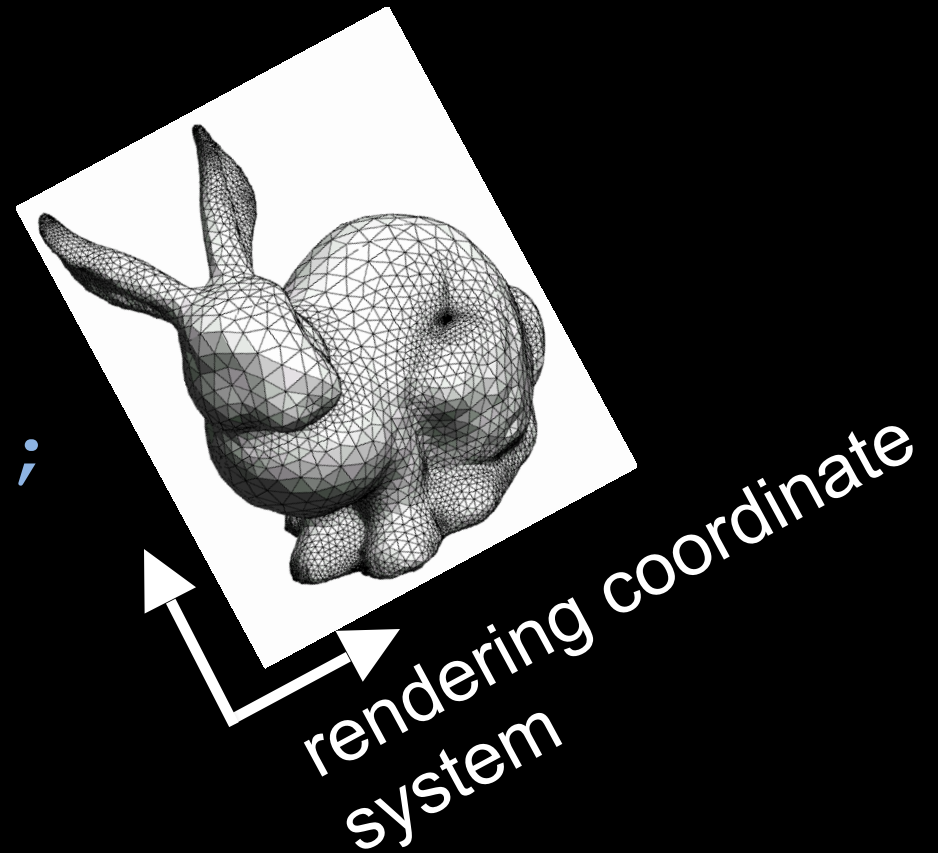
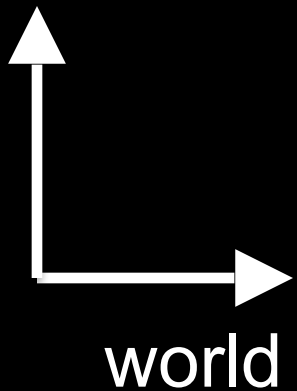
# The *rendering* coordinate system

```
Rotate(angle,  
       ax, ay, az);
```



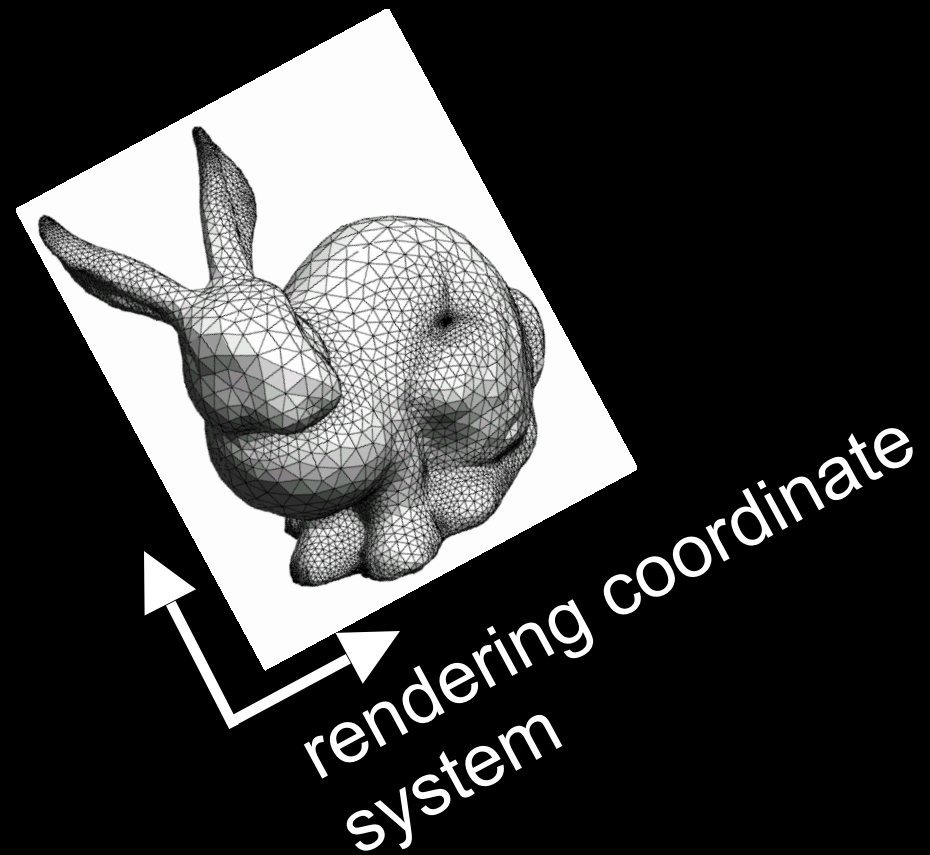
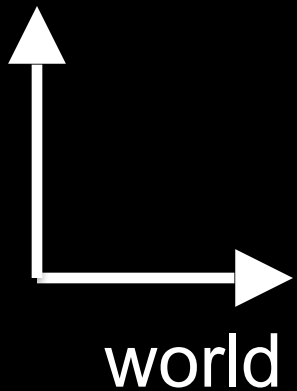
# The *rendering* coordinate system

```
Scale(sx, sy, sz);
```



# OpenGL pseudo-code

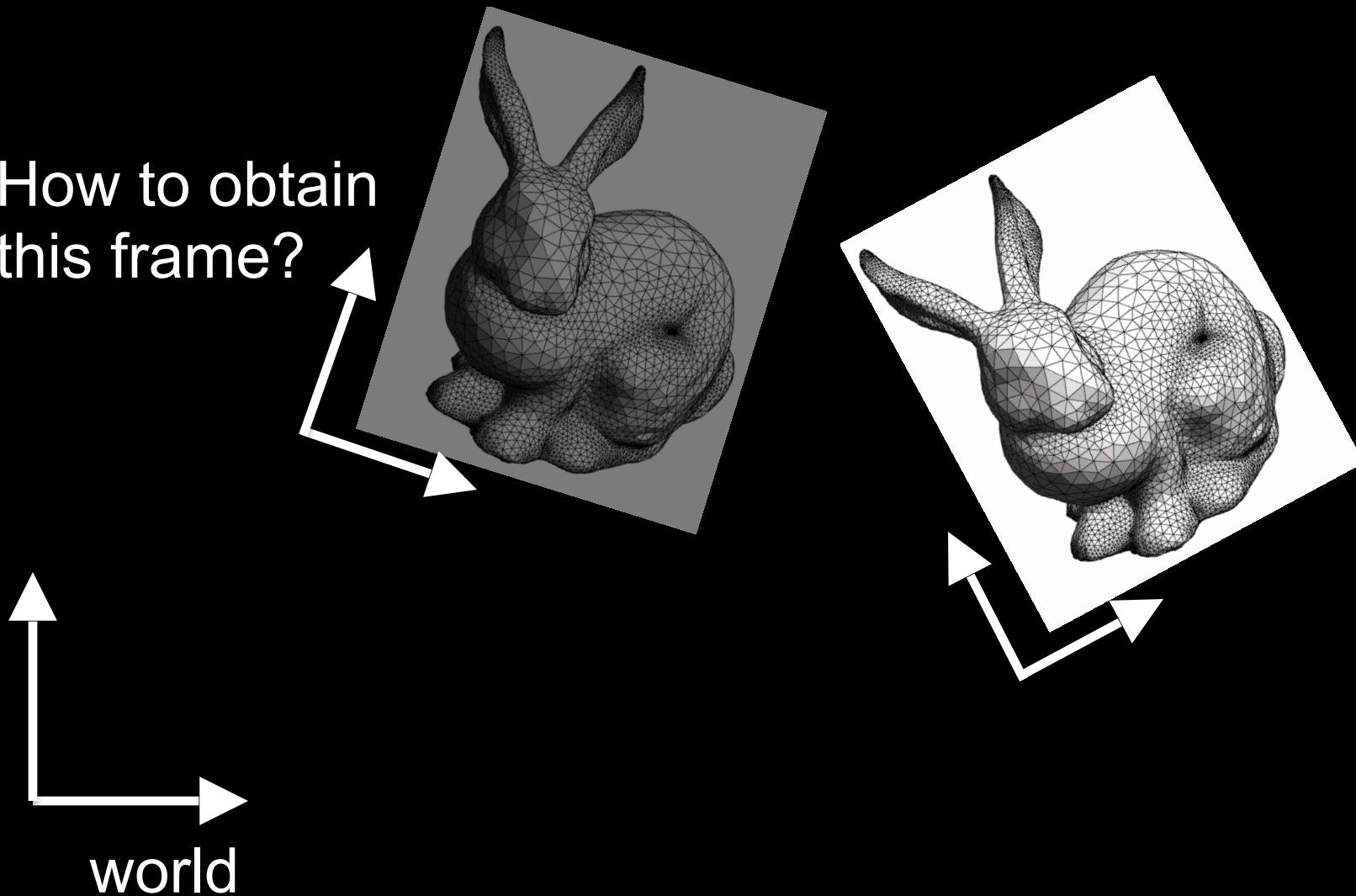
```
MatrixMode (ModelView);  
LoadIdentity();  
Translate(x, y, z);  
Rotate(angle, ax, ay, az);  
Scale(sx, sy, sz);  
glUniformMatrix4fv(...);  
renderBunny();
```





# Rendering more objects

How to obtain  
this frame?

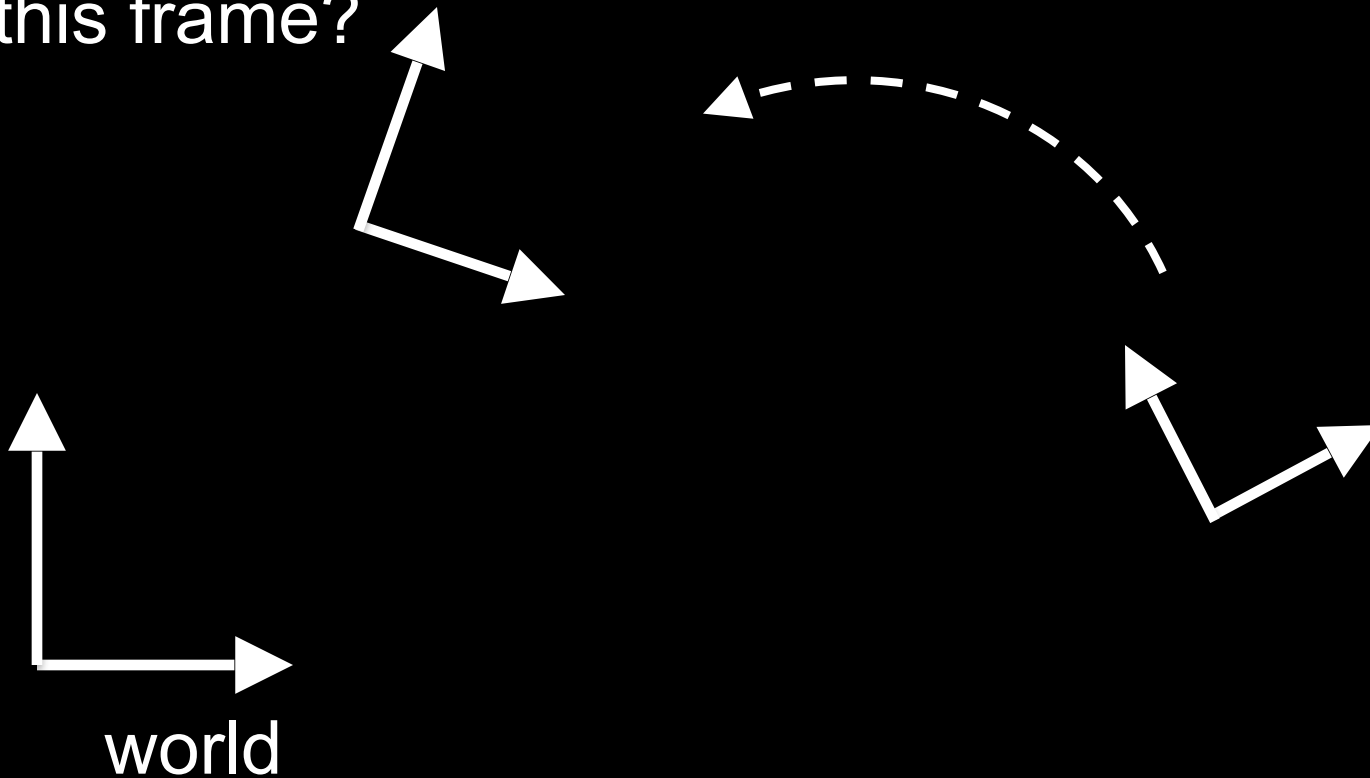


# Rendering more objects

## Solution 1:

Find `Translate(...)`, `Rotate(...)`,  
`Scale(...)`

How to obtain  
this frame?



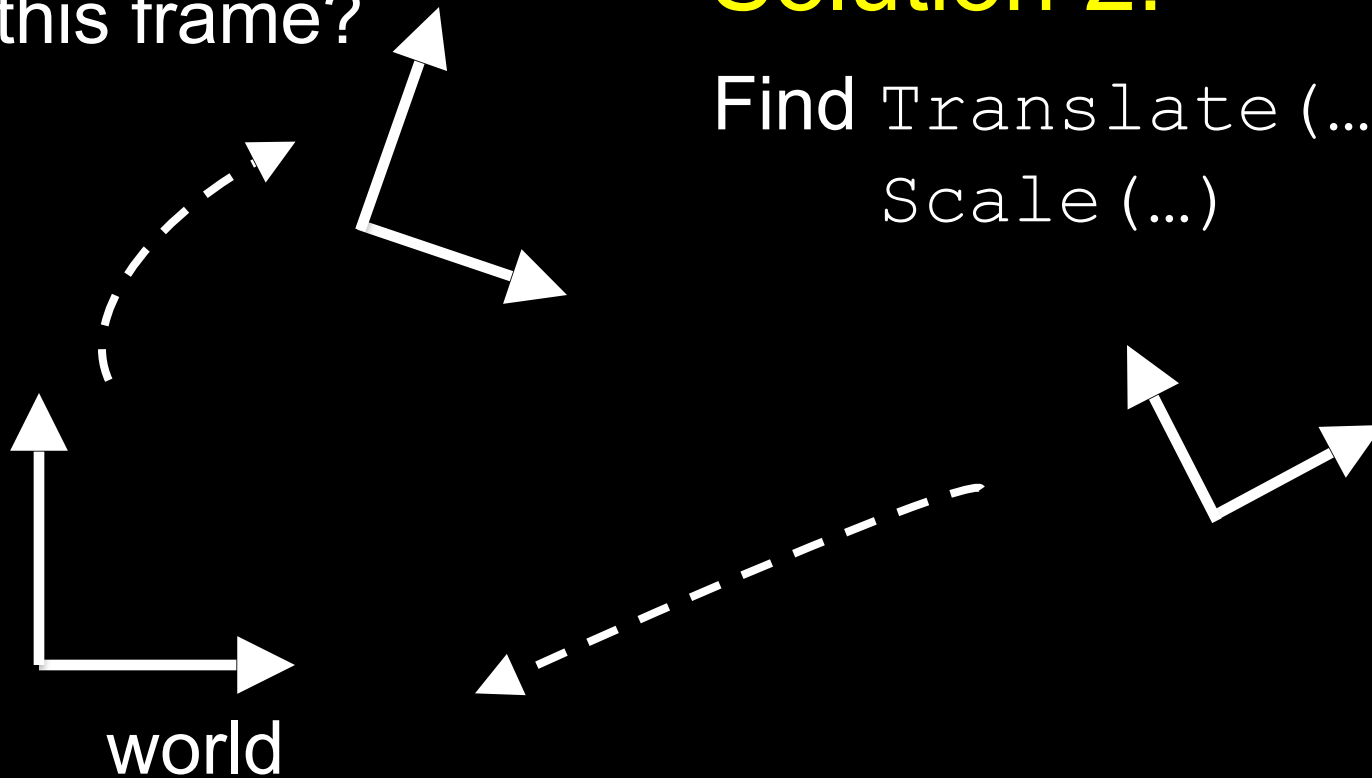


# Rendering more objects

How to obtain  
this frame?

## Solution 2:

Find Translate (...), Rotate (...),  
Scale (...)



# 3D Math Review

# 3D Math Review

(linear algebra with  $3 \times 3$   
and  $4 \times 4$  matrices)

# Scalars

- Scalars  $\alpha, \beta, \gamma$  from a *scalar field*
- Operations  $\alpha + \beta, \alpha \cdot \beta, 0, 1, -\alpha, ( )^{-1}$
- “Expected” laws apply
- Examples: rationals or reals with addition and multiplication



# Scalars

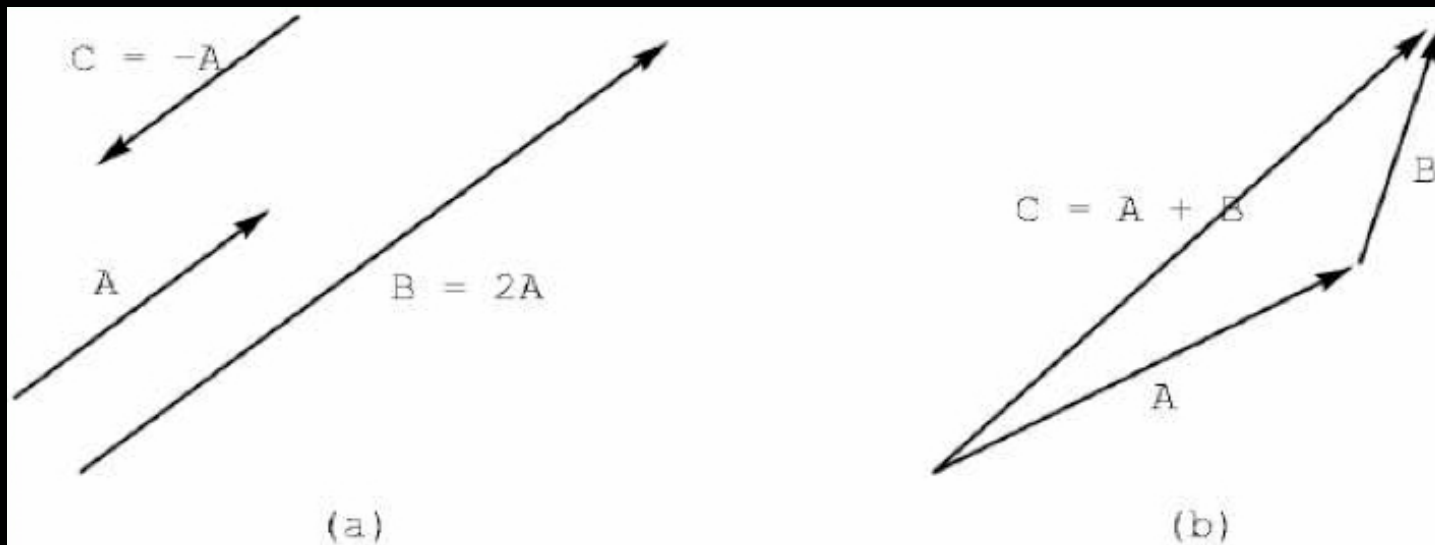
Scalars are numbers

`double, float, ...`

$$\alpha = 3$$

# Vectors

- Vectors  $u, v, w$  from a *vector space*
- Vector addition  $u + v$ , subtraction  $u - v$
- Zero vector  $0$
- Scalar multiplication  $\alpha v$



# Euclidean Space

- Vector space over real numbers

$$\mathbb{R}^n$$

mostly  $n=3$  or  $n=4$

# Euclidean Space

- Vector space over real numbers
- On CPU:
  - `float[3]`, `double[3]`, ...
- On GPU:
  - `vec3f`, ...

$$\mathbf{v} = (1, 5, 3)$$



# Euclidean Space

$$\mathbf{v} = (v_1, v_2, v_3)$$

$$\mathbf{w} = (w_1, w_2, w_3)$$

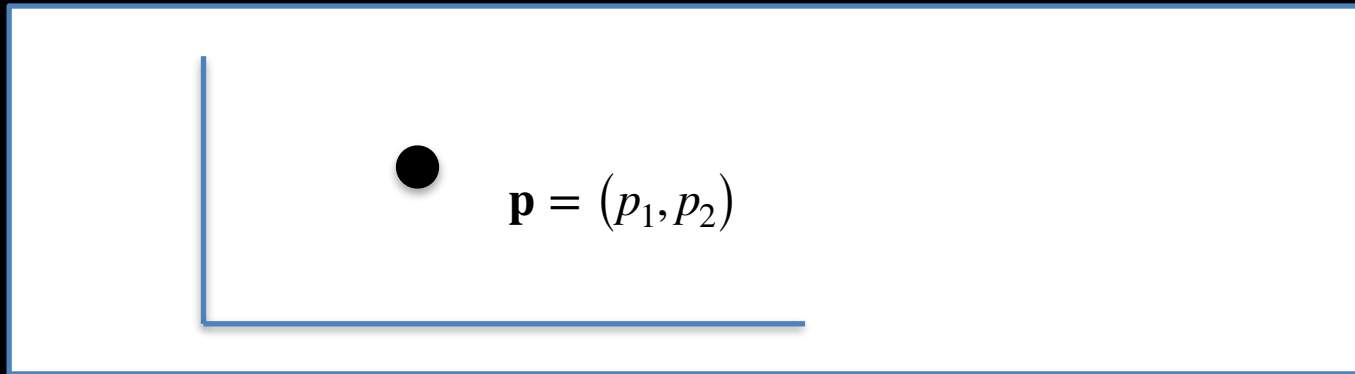
$$\mathbf{v} + \mathbf{w} = (v_1 + w_1, v_2 + w_2, v_3 + w_3)$$

# Euclidean Space

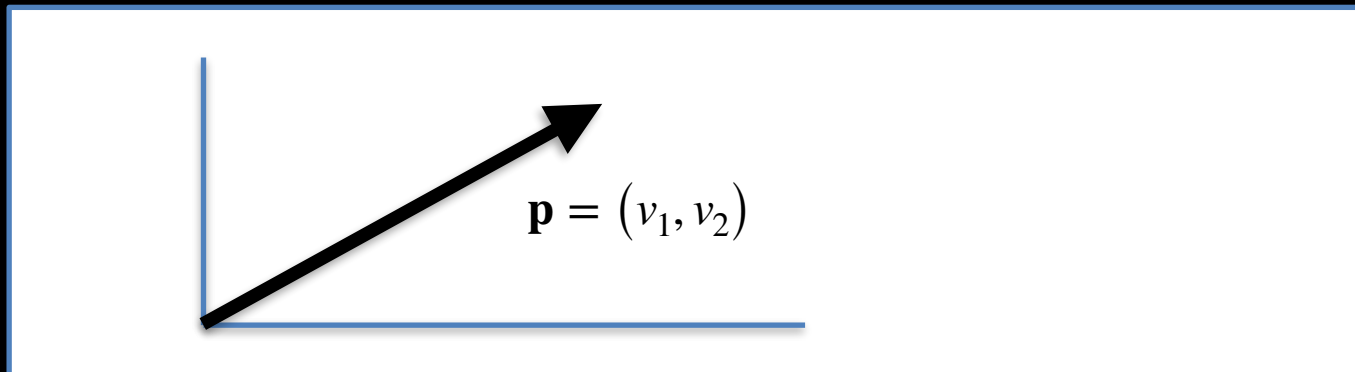
- Vector space over real numbers
- Three-dimensional in computer graphics
- Dot product:  $\alpha = u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3$
- $0 \cdot 0 = 0$
- $u, v$  are *orthogonal* if  $u \cdot v = 0$
- $|v|^2 = v \cdot v$  defines  $|v|$ , the *length* of  $v$
- If  $|v| = 1$ , the vector is *normal*

# Two interpretations of vectors

- Just a point in space



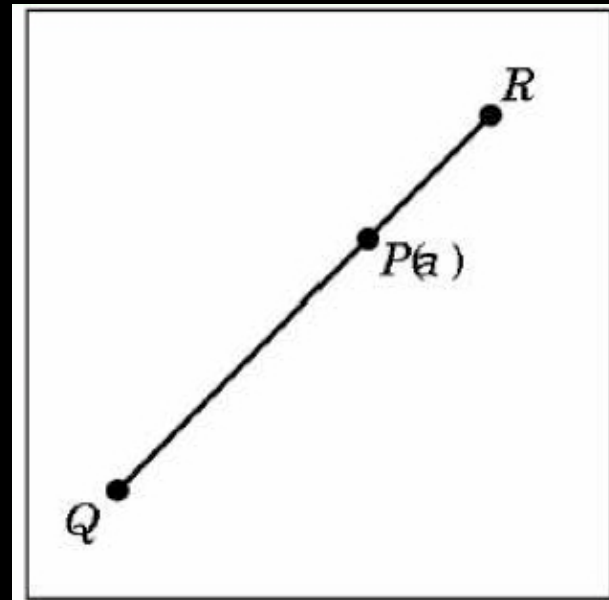
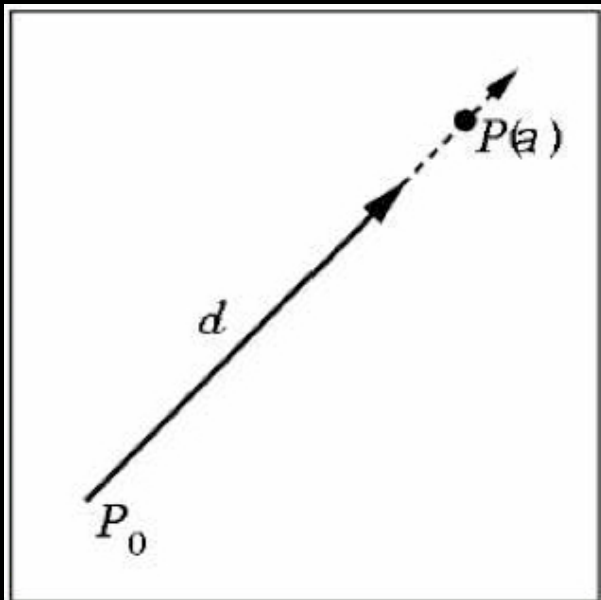
- An arrow pointing in a direction of a certain length



(some might say only the second is a vector,  
but in CG we represent both as vectors)

# Lines and Line Segments

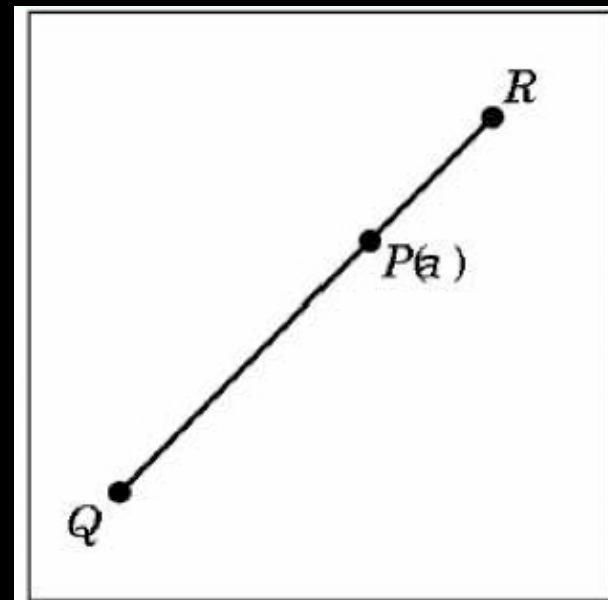
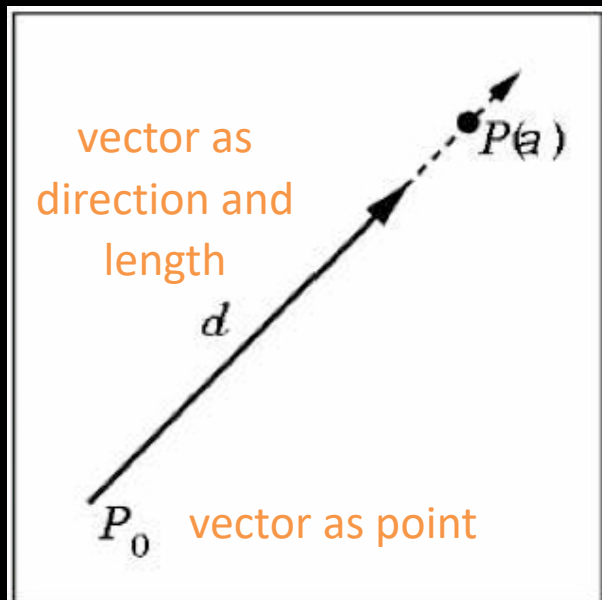
- Parametric form of line:  $P(\alpha) = P_0 + \alpha d$



- Line segment between  $Q$  and  $R$ :  
 $P(\alpha) = (1-\alpha) Q + \alpha R$  for  $0 \leq \alpha \leq 1$

# Lines and Line Segments

- Parametric form of line:  $P(\alpha) = P_0 + \alpha d$



- Line segment between  $Q$  and  $R$ :  
 $P(\alpha) = (1-\alpha) Q + \alpha R$  for  $0 \leq \alpha \leq 1$

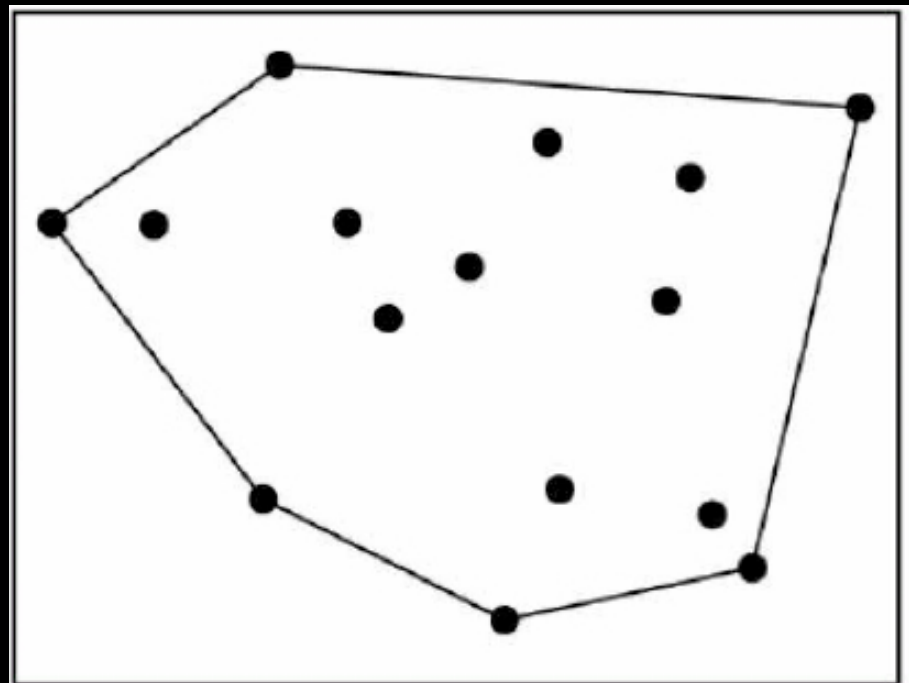
# Convex Hull

- Convex hull defined by

$$P = \alpha_1 P_1 + \dots + \alpha_n P_n$$

$$\text{for } \alpha_1 + \dots + \alpha_n = 1$$

$$\text{and } 0 \leq \alpha_i \leq 1, i = 1, \dots, n$$

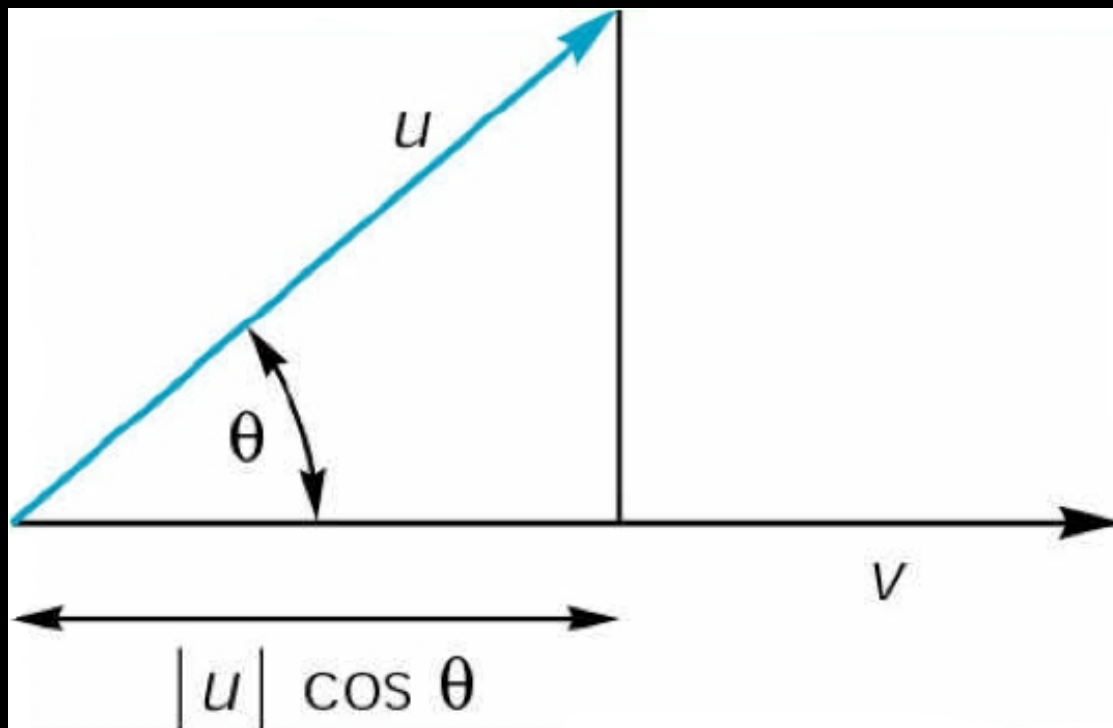


# Projection

- Dot product projects one vector onto another vector

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 = |u| |v| \cos(\theta)$$

$$\text{pr}_v u = (u \cdot v) v / |v|^2$$



# Orthogonal Projection

- Dot product projects one vector onto another vector

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 = |u| |v| \cos(\theta)$$

$$\text{pr}_v u = (u \cdot v) v / |v|^2$$

- Why is this called an orthogonal projection?



# Orthogonal Projection

- Dot product projects one vector onto another vector

$$u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 = |u| |v| \cos(\theta)$$

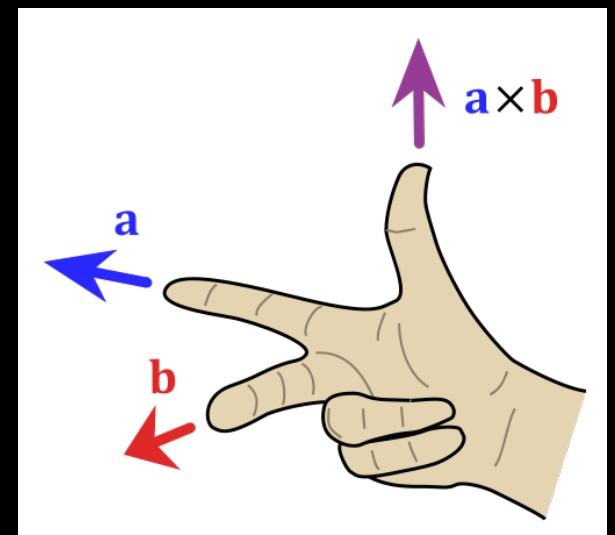
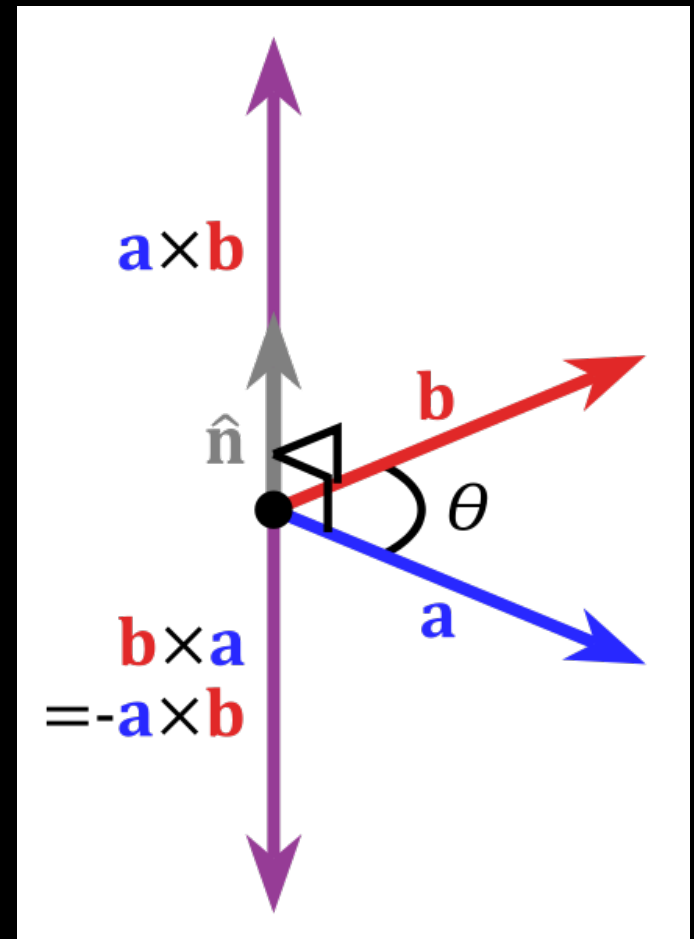
$$\text{pr}_v u = (u \cdot v) v / |v|^2$$

- Why is this called an orthogonal projection?
  - Because  $(u - \text{pr}_v u) \cdot v = (u \cdot v) - (u \cdot v) (v \cdot v) / |v|^2$   
 $= (u \cdot v) - (u \cdot v) = 0$

# Cross Product

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

- $|a \times b| = |a| |b| |\sin(\theta)|$
- Cross product is perpendicular to both  $a$  and  $b$
- Right-hand rule



# Plane

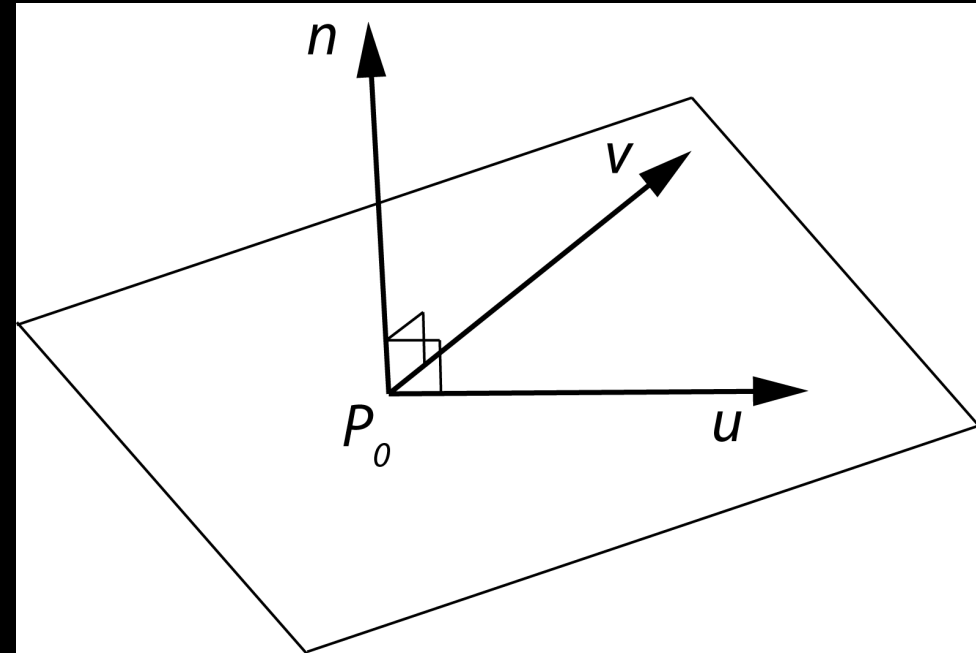
- Plane defined by point  $P_0$  and vectors  $u$  and  $v$

- $u$  and  $v$  should not be parallel

- Parametric form:

$$T(\alpha, \beta) = P_0 + \alpha u + \beta v$$

( $\alpha$  and  $\beta$  are scalars)



- $n = u \times v / |u \times v|$  is the normal

- $n \cdot (P - P_0) = 0$  if and only if  $P$  lies in plane

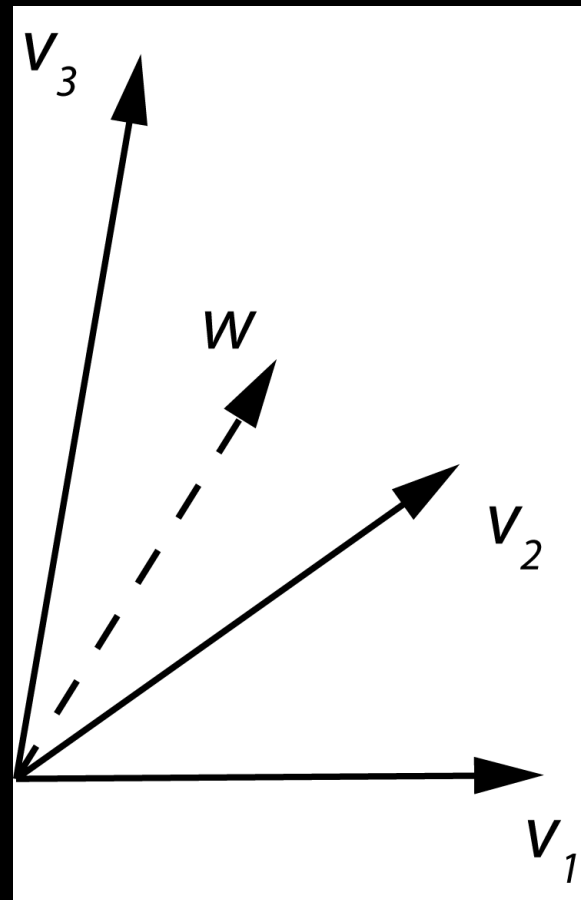
# Coordinate Systems

- Let  $v_1, v_2, v_3$  be three linearly independent vectors in a 3-dimensional vector space

- Can write *any* vector  $w$  as

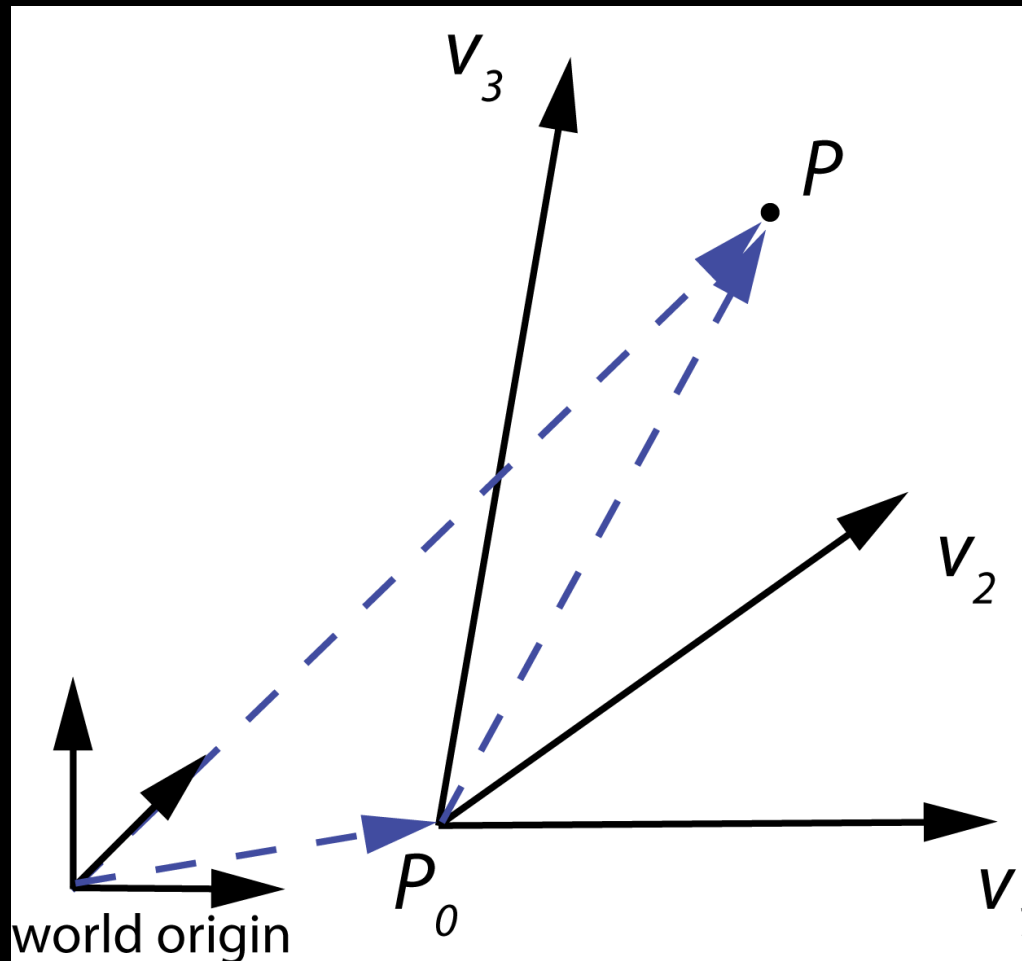
$$w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$$

for some scalars  $\alpha_1, \alpha_2, \alpha_3$

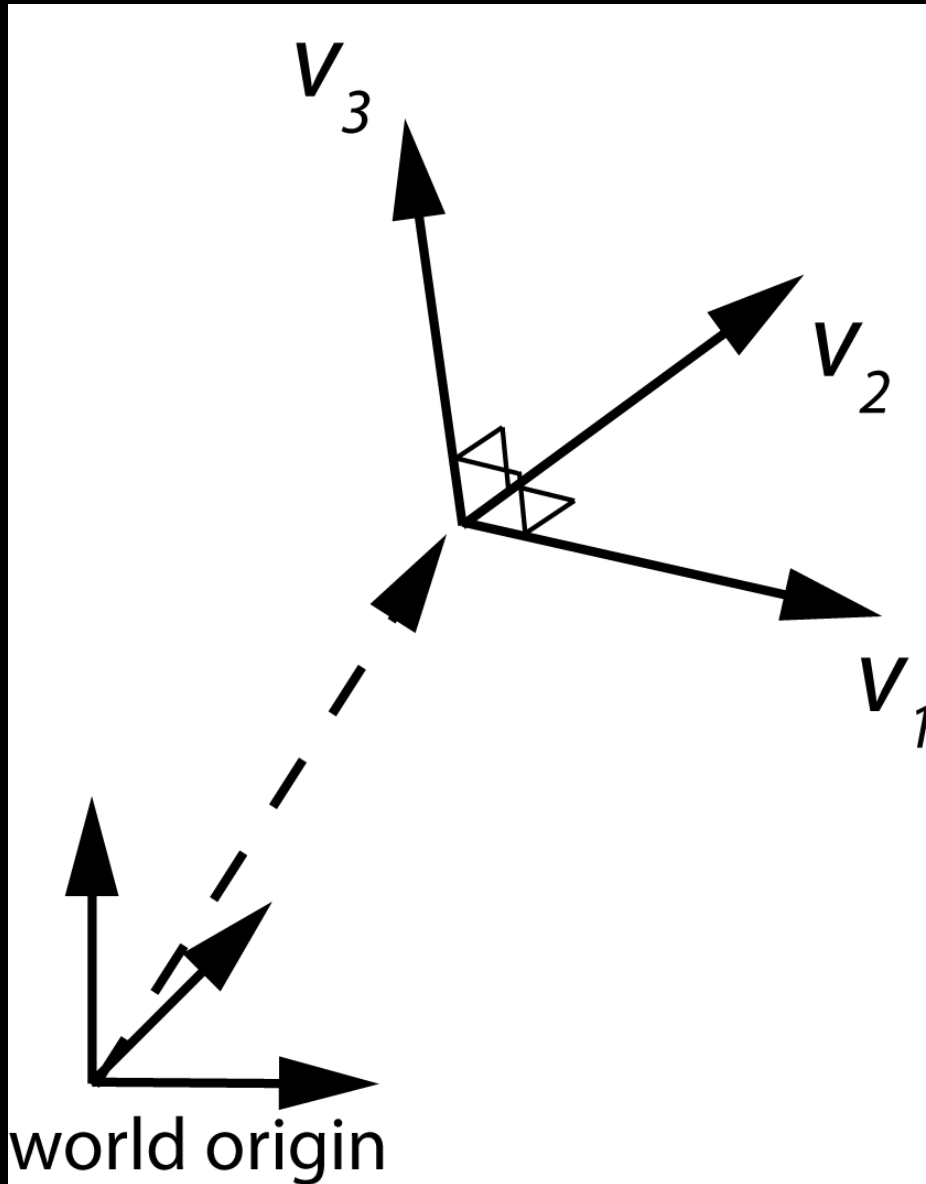


# Frames

- Frame = origin  $P_0$  + coordinate system
- Any point  $P = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$



# In Practice, Frames are Usually Orthonormal



$$v_1 \cdot v_2 = 0$$

$$v_2 \cdot v_3 = 0$$

$$v_1 \cdot v_3 = 0$$

$$\|v_1\| = \|v_2\| = \|v_3\| = 1$$

Orthonormality makes life easier

# Change of Coordinate System

- Bases  $\{u_1, u_2, u_3\}$  and  $\{v_1, v_2, v_3\}$
- Express basis vectors  $u_i$  in terms of  $v_j$

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$

$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$

$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$

- Represent in matrix form:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad M = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

# Matrices

- matrices represent linear transformations

$$\mathbf{a} = \mathbf{M}\mathbf{b}$$

$$\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

$$\mathbb{R}^{3 \times 3}$$

- On CPU:
  - `float[3][3]`, `double[3][3]`, ...
- On GPU:
  - `mat3f`, ...



# Matrices

- matrices can be added and multiplied by a scalar like a vector
- matrices can be multiplied with each other
  - composition of functions
- Determinant

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

$$\det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh - ceg - bdi - afh$$

# Matrices

- Inverse

$$A\mathbf{x} = \mathbf{y}$$

$$\mathbf{x} = A^{-1}\mathbf{y}$$

- Not all matrices can be inverted
  - Only if  $\det A \neq 0$
- Don't compute them explicitly - our helper libraries and your GPU can compute them.

# Matrices

- Orthogonal matrices:
  - $A = (a_1 \ a_2 \ a_3)$  is orthogonal if...

- $$A^T A = A A^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = I$$

- $a_1 \cdot a_2 = a_2 \cdot a_3 = a_1 \cdot a_3 = 0$  and  $\det A = 1$

# Matrices

- Can represent rotation, scaling, and reflection
  - Rotation:  $SO(3) = \{3 \times 3 \text{ orthogonal matrices with determinant } 1\}$
  - Scaling:  $Diag(3) = \{3 \times 3 \text{ matrices with positive entry on diagonal only}\}$
  - Reflection: Identity matrices where one diagonal element is -1
  - Rotation + Reflection:  $\{3 \times 3 \text{ matrices with } |\text{determinant}| = 1\}$
- Cannot represent translation
  - $u_{trans} = u + t$
  - There is no  $M$  such that  $u_{trans} = M u$

# In order to represent rotations, scales AND translations: Homogeneous Coordinates

- Augment  $[\alpha_1 \ \alpha_2 \ \alpha_3]^T$  by adding a fourth component (1):

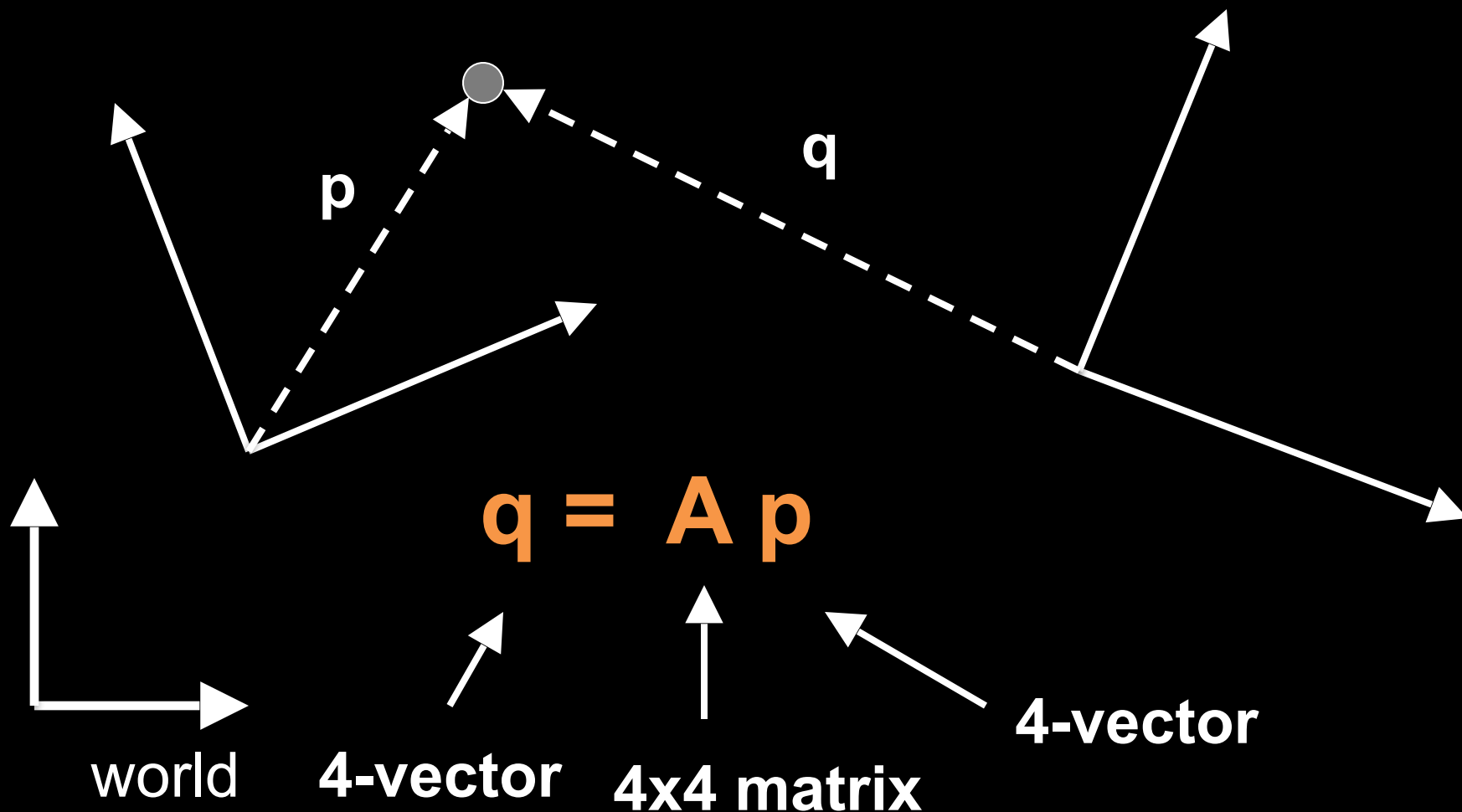
$$\mathbf{p} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 1]^T$$

- Homogeneous property:

$$\mathbf{p} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 1]^T = [\beta\alpha_1 \ \beta\alpha_2 \ \beta\alpha_3 \ \beta]^T ,$$

for any scalar  $\beta \neq 0$

# Homogeneous coordinates are transformed by 4x4 matrices



# Affine Transformations (4x4 matrices)

- Translation
- Rotation
- Scaling
- Any composition of the above
- Later: projective (perspective) transformations
  - Also expressible as 4 x 4 matrices!

# Translation

- $\mathbf{q} = \mathbf{p} + \mathbf{d}$  where  $\mathbf{d} = [\alpha_x \ \alpha_y \ \alpha_z \ 0]^\top$
- $\mathbf{p} = [x \ y \ z \ 1]^\top$
- $\mathbf{q} = [x' \ y' \ z' \ 1]^\top$
- Express in matrix form  $\mathbf{q} = \mathbf{T} \mathbf{p}$  and solve for  $\mathbf{T}$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



# Scaling

- $x' = \beta_x x$
- $y' = \beta_y y$
- $z' = \beta_z z$
- Express as  $\mathbf{q} = \mathbf{S} \mathbf{p}$  and solve for  $\mathbf{S}$

$$\mathbf{S} = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Rotation in 2 Dimensions

- Rotation by  $\theta$  about the origin
- $x' = x \cos \theta - y \sin \theta$
- $y' = x \sin \theta + y \cos \theta$
- Express in matrix form:

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

- Note that the determinant is 1

# Rotation in 3 Dimensions

- Orthogonal matrices:

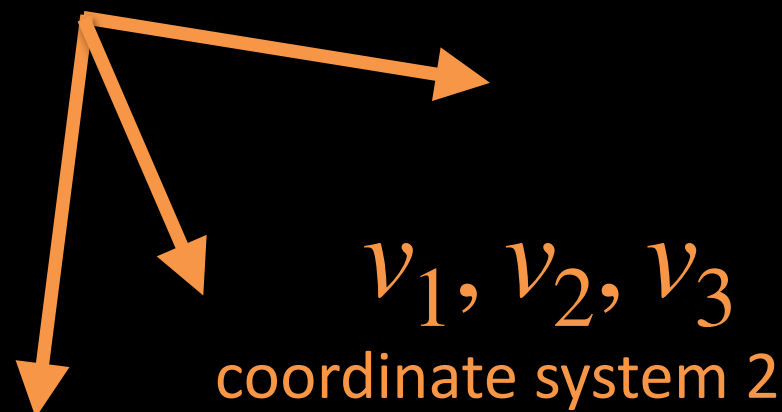
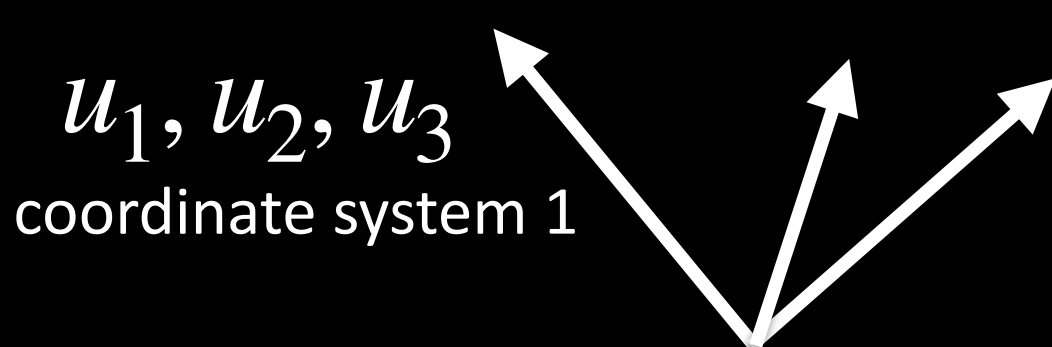
$$RR^T = R^T R = I$$
$$\det(R) = 1$$

- As a 4x4 homogeneous matrix:

$$A = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

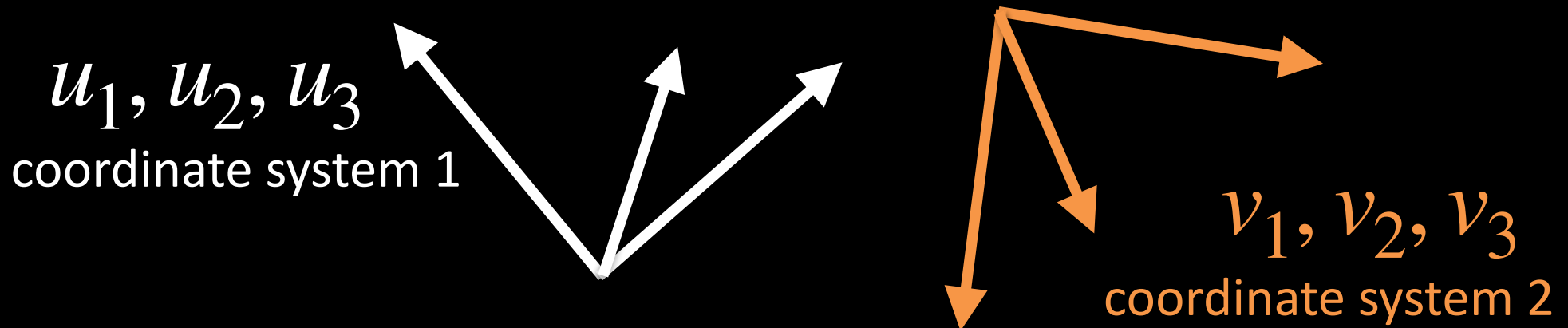
# Rotation in 3 Dimensions

- How do we rotate between two orthonormal coordinate systems?



# Rotation in 3 Dimensions

- How do we rotate between two orthonormal coordinate systems?



- $U = \begin{pmatrix} u_1 & u_2 & u_3 \end{pmatrix}$  transforms system 1 to canonical system
- $V = \begin{pmatrix} v_1 & v_2 & v_3 \end{pmatrix}$  transforms system 2 to canonical system
- To transform  $\mathbf{x}$  from system 1 to system 2,  
 $\mathbf{x} \mapsto V^{-1}U\mathbf{x}$

# Affine Matrices are Composed by Matrix Multiplication

- $\mathbf{A} = \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3$
- Applied from right to left
- $\mathbf{A} \mathbf{p} = (\mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3) \mathbf{p} = \mathbf{A}_1 (\mathbf{A}_2 (\mathbf{A}_3 \mathbf{p}))$
- Compatibility mode:  
When calling `glTranslate3f`, `glRotatef`, or `glScalef`, OpenGL forms the corresponding 4x4 matrix, and multiplies the current modelview matrix with it.

# Summary

- OpenGL Transformation Matrices
- Vector Spaces
- Frames
- Homogeneous Coordinates
- Transformation Matrices