Camera Models

Spring 2025

Daniel G. Aliaga
Department of Computer Science
Purdue University

Typical OpenGL Matrices

- Projection Matrix
  - Defines the projection process: perspective, orthographic, etc.
- ModelView Matrix (or View Matrix)
  - Defines where is the camera
- Model Matrices
  - Applied to geometry/model to define scene objects
- Texture Matrix
  - Is applied to the “texture” (more on this later)

Transformations

- Most popular transformations in graphics
  - Translation
  - Rotation
  - Scale
  - Projection
- In order to use a single matrix for all, we use homogeneous coordinates (we talked about this already)

3D Transformations

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

Identity

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Scale

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & tx \\ 0 & 1 & 0 & ty \\ 0 & 0 & 1 & tz \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Mirror over X axis

3D Transformations

Rotate around Z axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta & 0 & 0 \\ \sin\Theta & \cos\Theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Rotate around Y axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\Theta & 0 & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

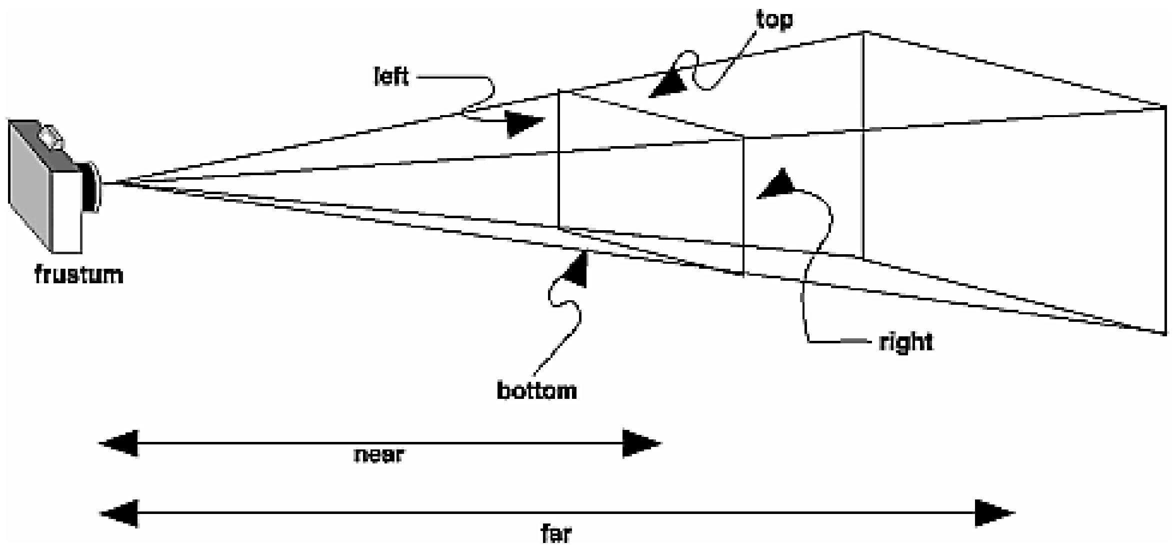
Rotate around X axis:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\Theta & -\sin\Theta & 0 \\ 0 & \sin\Theta & \cos\Theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

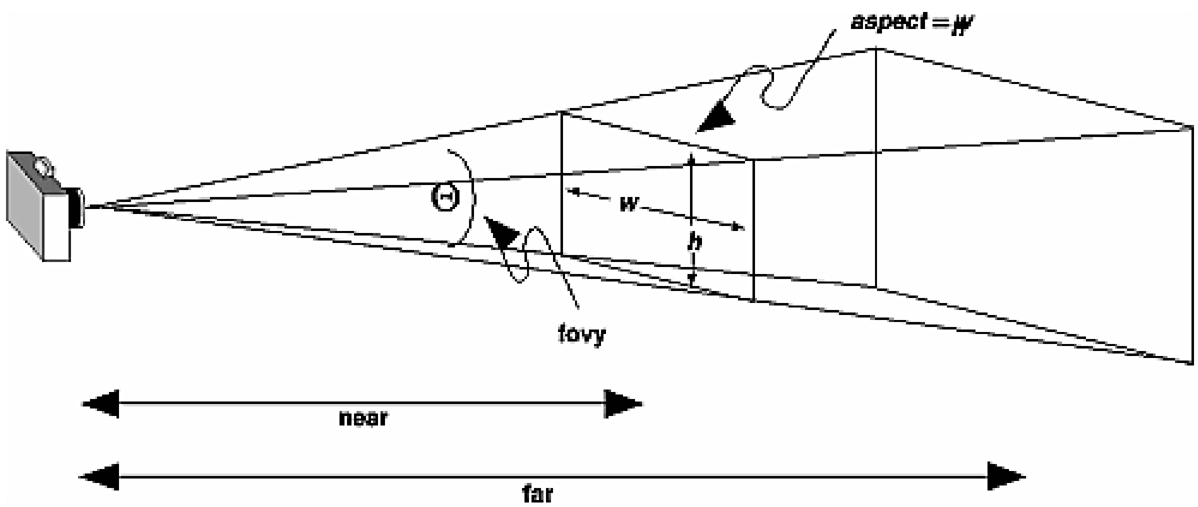
Perspective projection

Projection Transformations



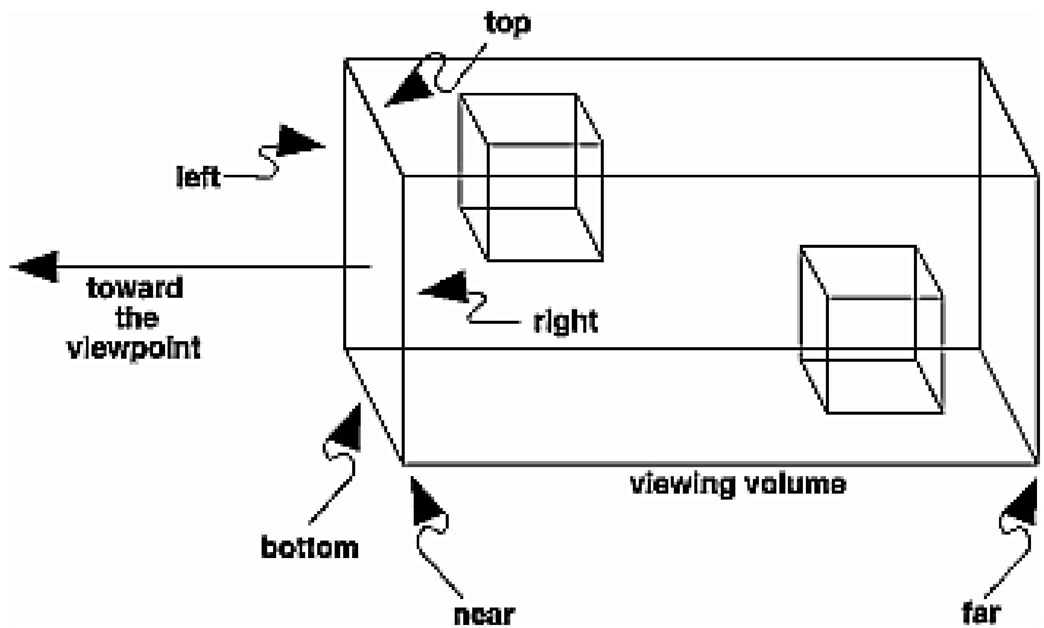

```
void glFrustum(GLdouble left, GLdouble right, GLdouble
  bottom, GLdouble top, GLdouble near, GLdouble far);
```

Projection Transformations




```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble
  near, GLdouble far);
```

Projection Transformations



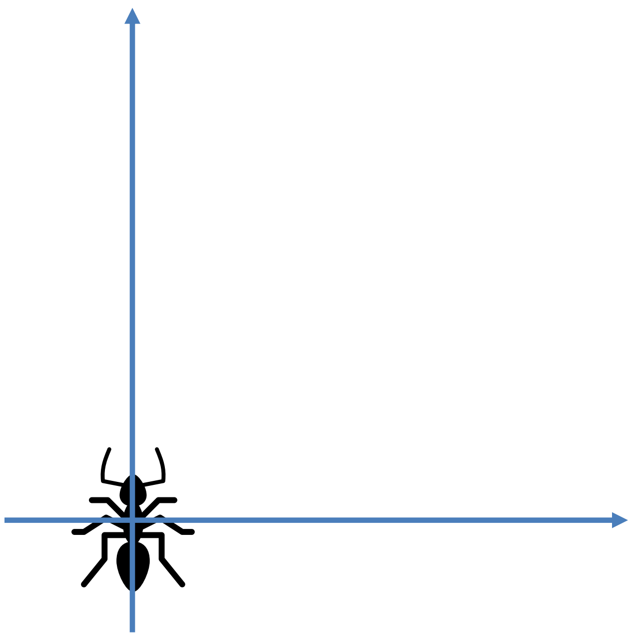

```
void glOrtho(GLdouble left, GLdouble right, GLdouble
  bottom,
  GLdouble top, GLdouble near, GLdouble far);

void gluOrtho2D(GLdouble left, GLdouble right,
  GLdouble bottom, GLdouble top);
```

View/Model Transformations

- The order of operations matters!
- How to rotate CW 90^o?
- Solution?

```
Rotate(90)
```

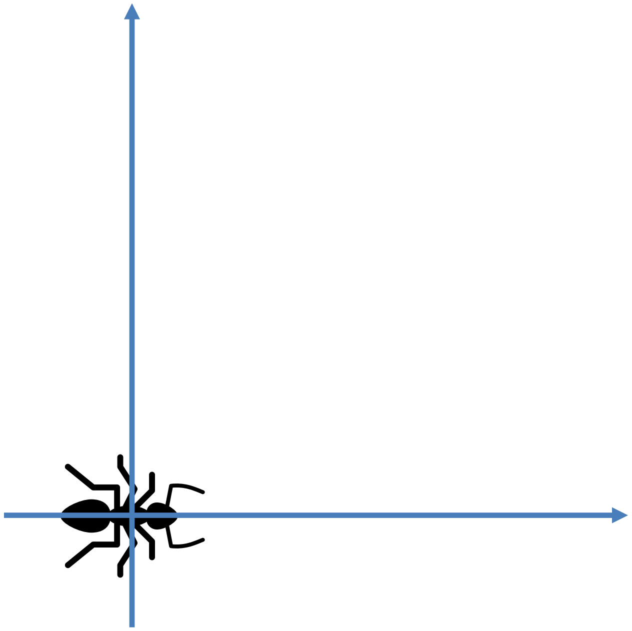


```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CW 90°?
- Solution?

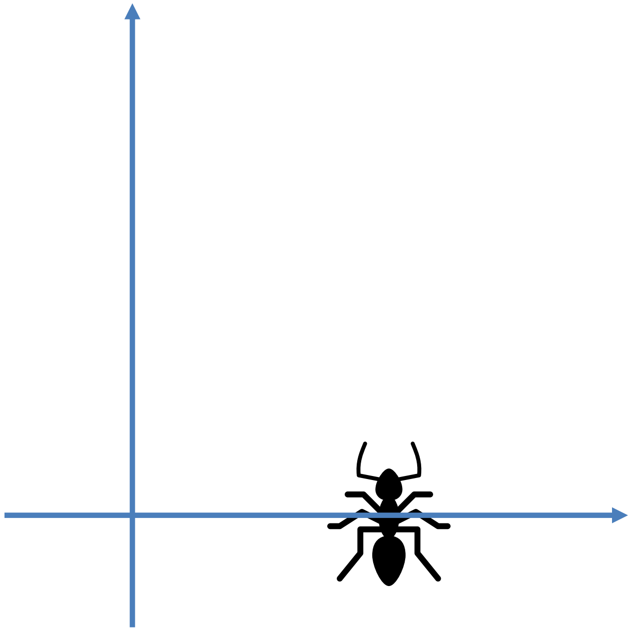
```
Rotate(90)
```



```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90°?
- Solution?

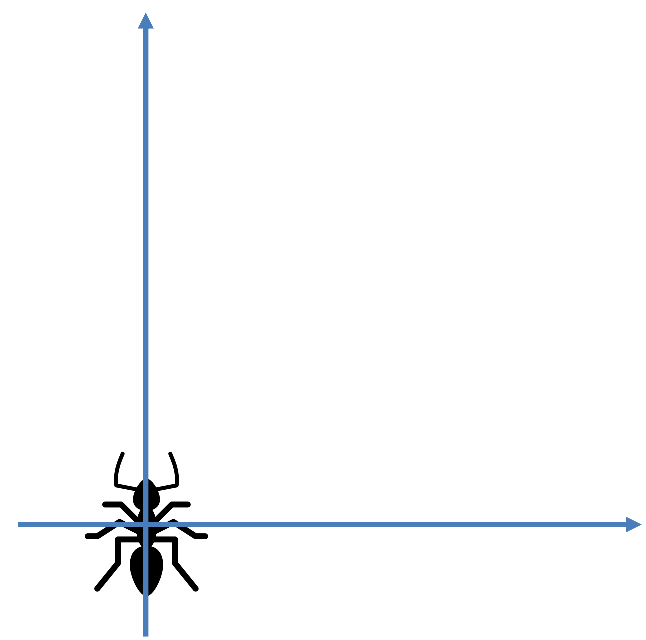


```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90^o?
- Solution?

```
Translate(-a)
```

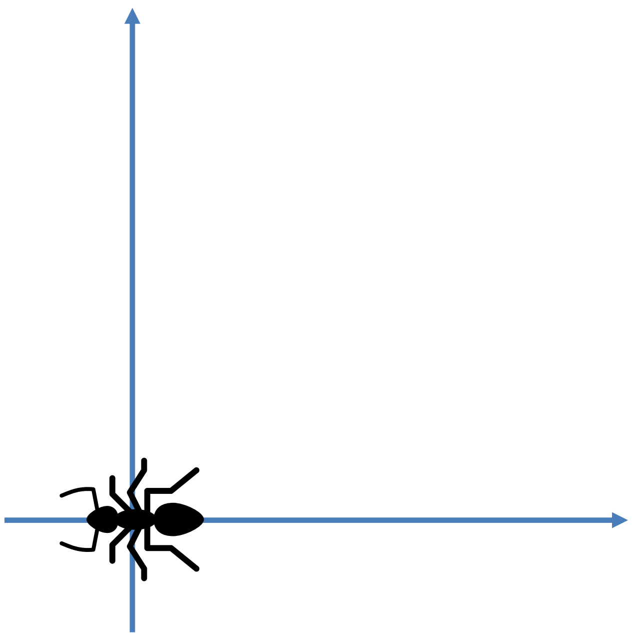


```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90°?
- Solution?

```
Translate(-a)
Rotate(-90)
```

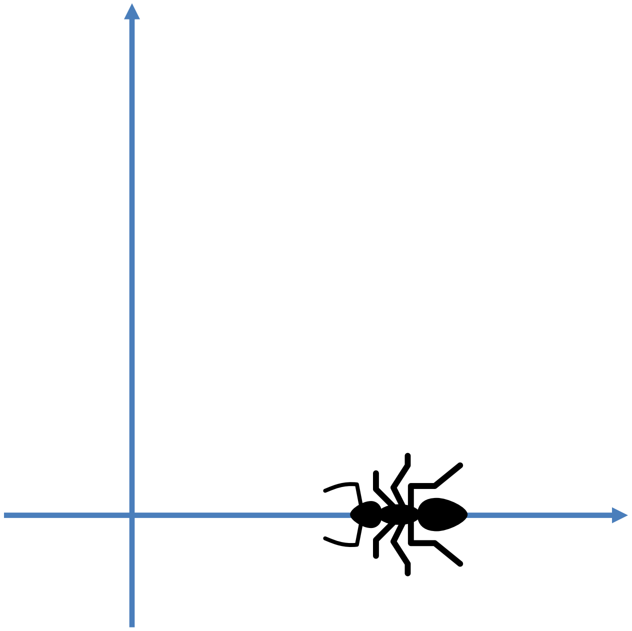


```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90^o?
- Solution?

```
Translate(-a)
Rotate(-90)
Translate(a)
```

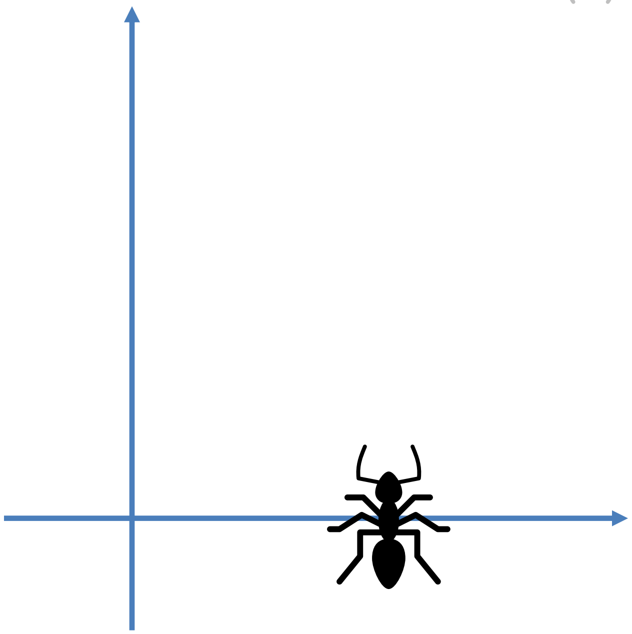


```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90^o?
- What if I rotate first?

```
Previous solution:
  Translate(-a)
  Rotate(-90)
  Translate(a)
```



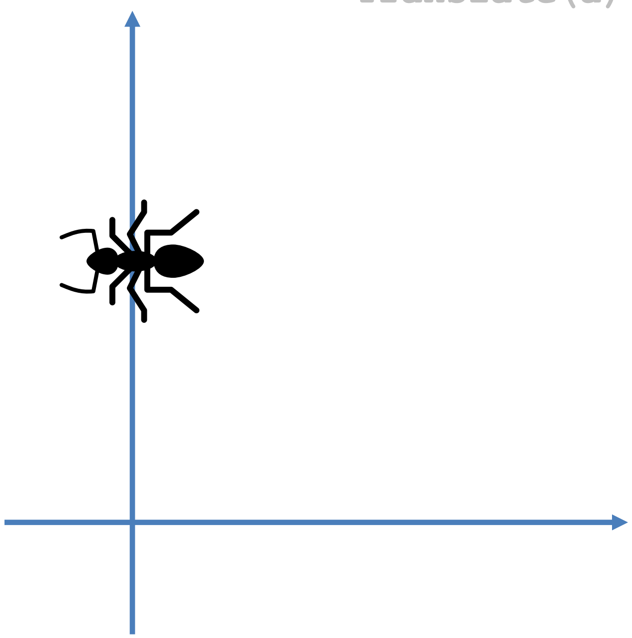
```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90°?
- What if I rotate first?

Previous solution:
Translate(-a)
Rotate(-90)
Translate(a)

Rotate(-90)



Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t

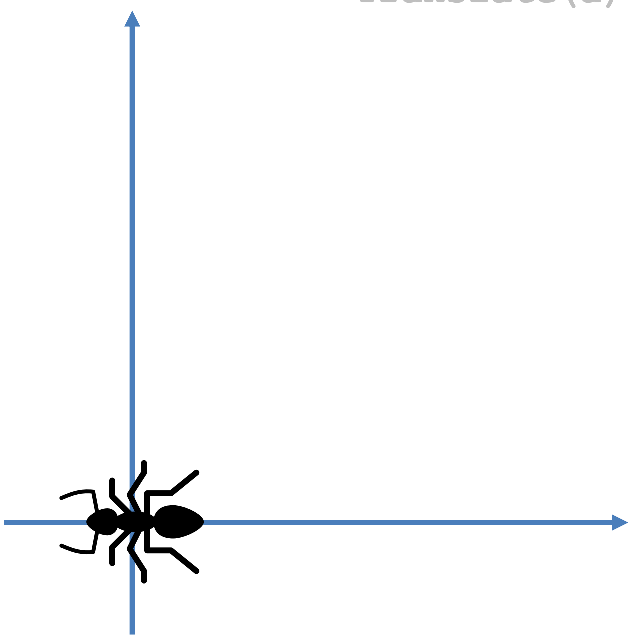
View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90°?
- What if I rotate first?

Previous solution:
Translate(-a)
Rotate(-90)
Translate(a)

```
Rotate(-90)
Translate(-a)
```

[assuming a was updated to new position]



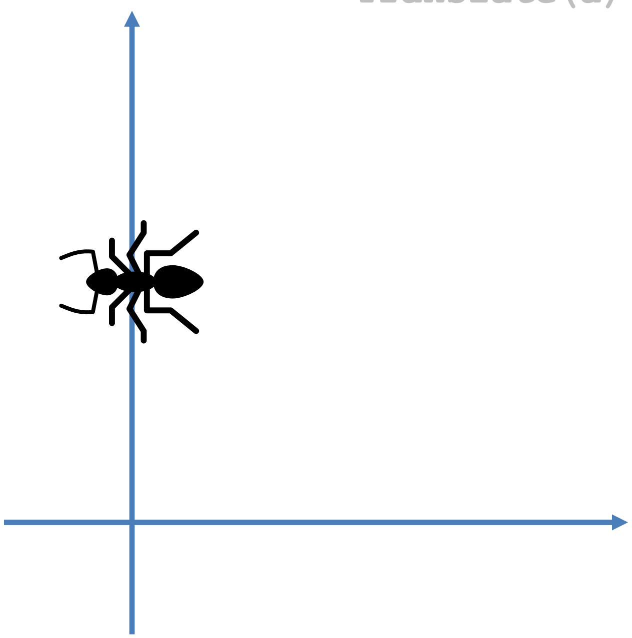
```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- The order of operations matters!
- How to rotate CCW 90^o?
- What if I rotate first?

```
Previous solution:
  Translate(-a)
  Rotate(-90)
  Translate(a)
```

```
Rotate(-90)
Translate(-a)
Translate(a)
```



```
Ant position = a
Rotate(d): rotate CW by d degrees
Translate(t): translate by vector t
```

View/Model Transformations

- In matrix form:

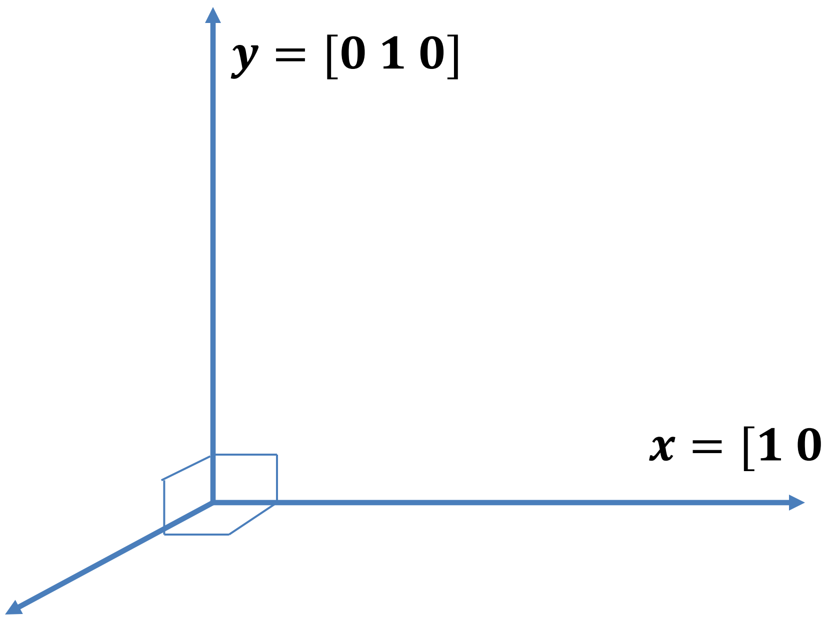
```
rMat = RotateMat(-90)
inv_tMat = TranslateMat(-a)
tMat = TranslateMat(a)

p′ = tMat * rMat * inv_tMat * p
(rotates points p of the ant “about itself”)


p′ = tMat * inv_tMat * rMat * p = rMat * p
(rotates points p of the ant around the origin
```

Change of Basis Transformation

- Standard basis:

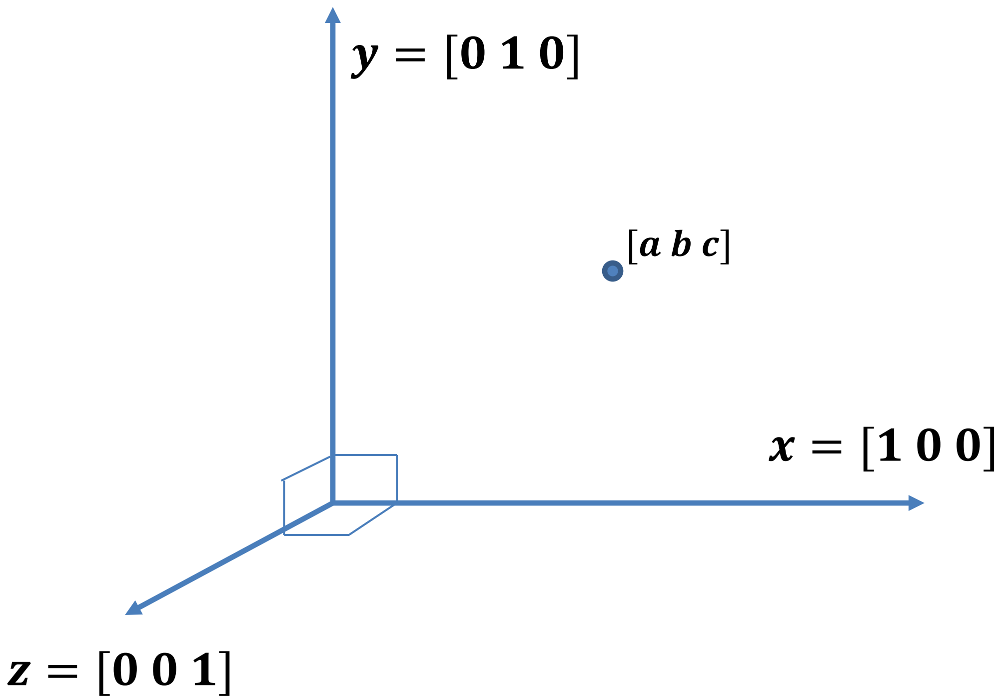


$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

x^T y^T z^T

Change of Basis Transformation

- Standard basis:



$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

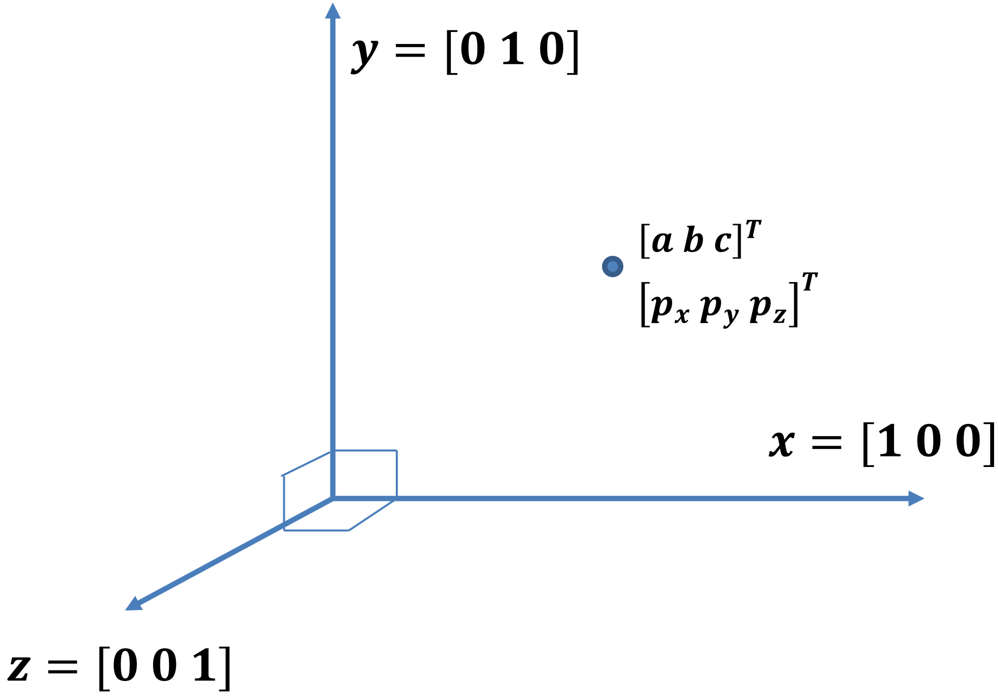
x^T y^T z^T

Where is point $[a\ b\ c]$ in basis S?

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = S \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Change of Basis Transformation

- Standard basis:



$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

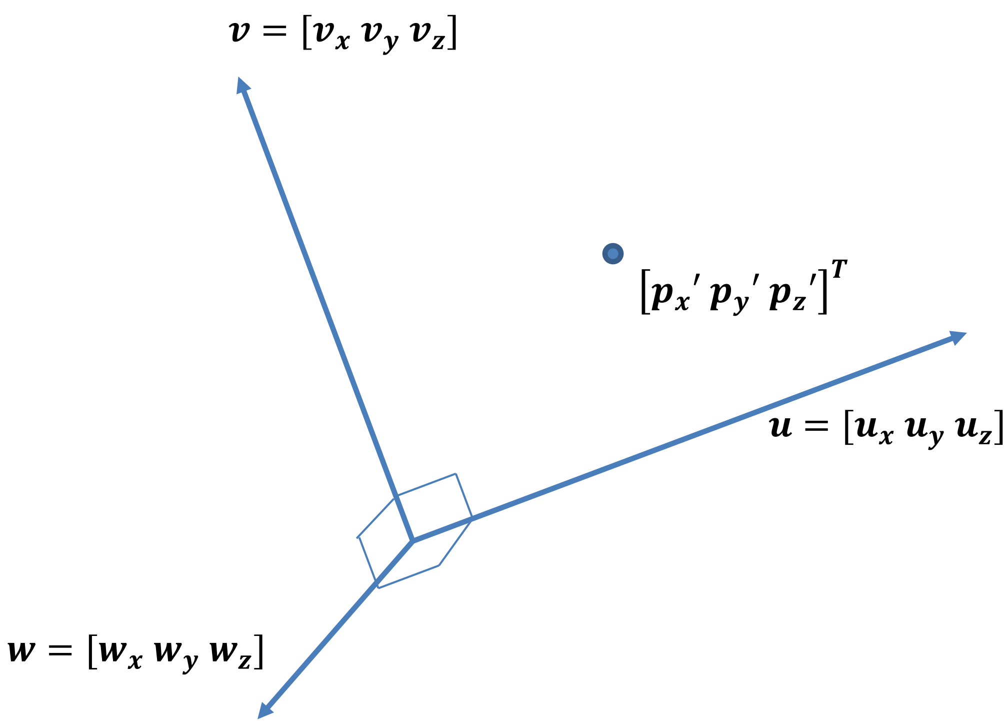
$x^T \quad y^T \quad z^T$

Where is point $[a\ b\ c]$ in basis S?

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = S \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Change of Basis Transformation

- Basis B:



$$B = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

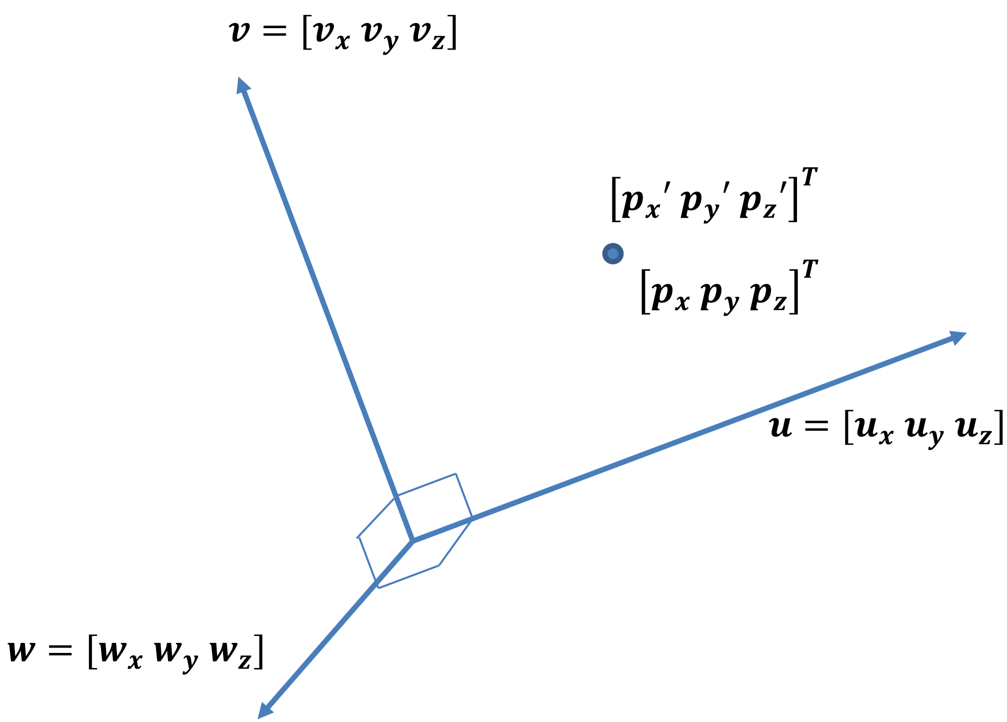
What is B?

Where is point $[p_x'\ p_y'\ p_z']$ **from basis B?**

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = B \begin{bmatrix} p_x' \\ p_y' \\ p_z' \end{bmatrix}$$

Change of Basis Transformation

- Basis B:



$$B = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}$$

$u^T \quad v^T \quad w^T$

Where is point $[p'_x \; p'_y \; p'_z]$ in basis B?

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = B \begin{bmatrix} p_x{}' \\ p_y{}' \\ p_z{}' \end{bmatrix}$$

Change of Basis Transformation

- In matrix form:

```
// change p′ from basis b to standard basis
bMat = makeBasisMat(u,v,w)
p′ = position(1,1,1)
p = bMat * p′

// change from standard to basis B
p′ = inverse(bMat) * p
```

Change of Basis Transformation

- What else is this change of basis useful for?
  - Rotating to an arbitrary basis
  - "I was in basis frame (x,y,z) and now I want to rotate to be basis frame (u,v,w)"

Change of Basis Transformation

- Recall we did "inverse(bMat)"
- What is the inverse of matrix?

$$B^{-1}B = I$$

- A nice property:
  - If B is formed by orthogonal basis vectors, then its inverse is simply:

$$B^{-1} = B^T = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix}$$