Introduction to Computer Graphics

Transformations

葉奕成 I-Cheng (Garrett) Yeh

Objectives

- Introduce concepts such as dimension and basis
- Introduce coordinate systems for representing vectors spaces and frames for representing affine spaces
- Discuss change of frames and bases

Linear Independence

- A set of vectors v_1, v_2, …, v_n is *linearly independent* if the equation

  $$\alpha_1 v_1 + \alpha_2 v_2 + .. \alpha_n v_n = 0$$

  can only be satisfied by

  $$\alpha_1 = \alpha_2 = \ldots = 0$$

- If a set of vectors is linearly *independent*, we cannot represent one in terms of the others
- If a set of vectors is linearly *dependent*, at least one can be written in terms of the others

Dimension

- In a vector space, the maximum number of linearly independent vectors is fixed and is called the *dimension* of the space
- In an *n*-dimensional space, any set of n linearly independent vectors *form* a *basis* for the space
- Given a basis $v_1, v_2, \ldots, v_n$, any vector v can be written as

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$$

where the $\{\alpha_i\}$ are unique

Representation

- Until now we have been able to work with geometric entities without using any frame of reference, such a coordinate system
- Need a frame of reference to relate points and objects to our physical world.
  - For example, where is a point?
    We can't answer it without a reference system
- Introduce…
  - *World coordinates*
  - *Camera coordinates*

Coordinate Systems

- Consider a basis $v_1, v_2, \ldots, v_n$
- A vector is written $v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
- The list of scalars $\{\alpha_1, \alpha_2, \ldots \alpha_n\}$ is the *representation of v, with respect to the given basis*
- We can write the representation as a row or column array of scalars

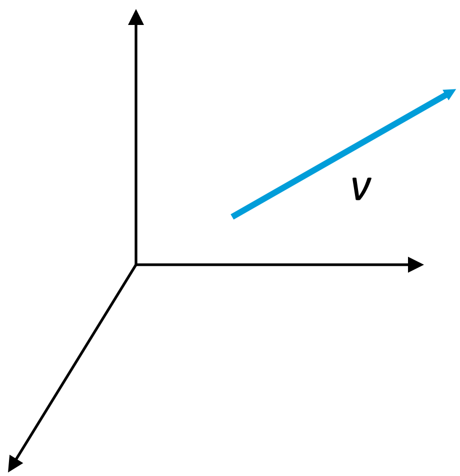
$$A = [\alpha_1 \ \alpha_2 \ \ldots \ \alpha_n]^T = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \ldots \\ \alpha_n \end{bmatrix}$$

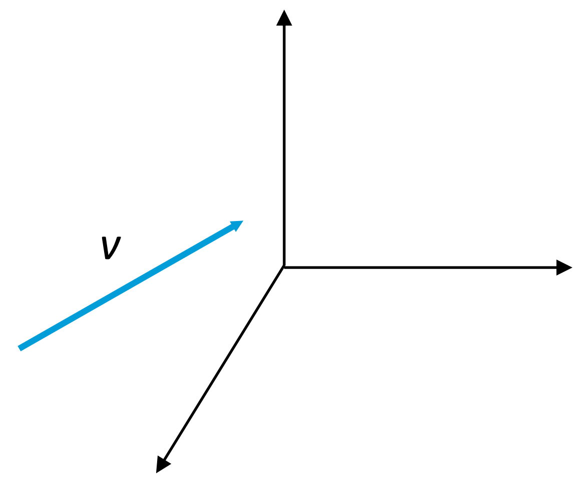
Example

- $V = 2\,v_1 + 3\,v_2 - 4\,v_3$
- $A = [\,2 \;\; 3 \;\; -4\,]$
  - Note that this representation is with respect to a particular basis
- For example, in OpenGL we start by representing vectors using the *world basis* but later the system needs a representation in terms of the *camera* or *eye basis*

Coordinate Systems

- Which is correct?

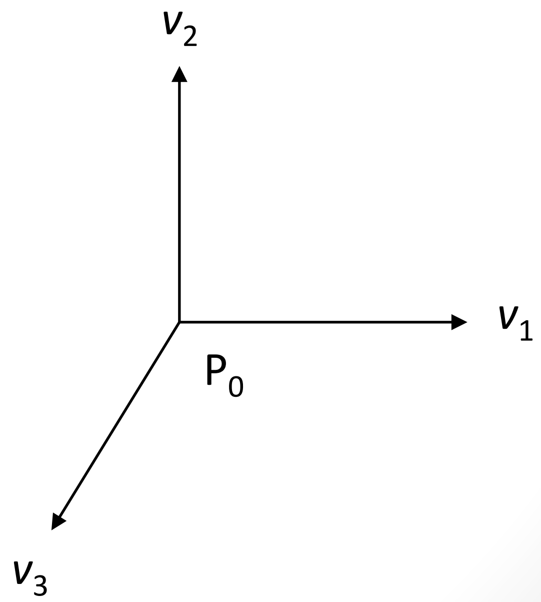




- Both of them are correct, because vectors have no fixed location.

Frame of reference

- Coordinate System is insufficient to present points
- If we work in an affine space we can add a single point, the *origin*, to the basis vectors to form a *frame*



Representation in a Frame

- Frame determined by $(P_0, v_1, v_2, \ldots, v_n)$

- Within this frame, every vector can be written as

  $v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots. + \alpha_n v_n$

- Every point can be written as

  $p = P_0 + \beta_1 v_1 + \beta_2 v_2 + \ldots. + \beta_n v_n$

Homogeneous Coordinates

The general form of three dimensional homogeneous coordinates is $\mathbf{p}=[x\ y\ z\ w]^T$

We go back to a three dimensional point (for $w \neq 0$) by

$x \leftarrow x/w$

$y \leftarrow y/w$

$z \leftarrow z/w$

*If $w = 0$, the representation is that of a vector

Note that homogeneous coordinates replaces points in three dimensions by lines through the origin in four dimensions

Homogeneous Coordinates and Computer Graphics

- Homogeneous coordinates are key to all computer graphics systems
  - All standard transformations (rotation, translation, scaling) can be implemented by matrix multiplications with 4 x 4 matrices
  - Hardware pipeline works with 4 dimensional representations
  - *For *orthographic* viewing, we can maintain $\mathrm{w} = 0$ for vectors and $\mathrm{w} = 1$ for points
  - *For *perspective* we need a *perspective division*

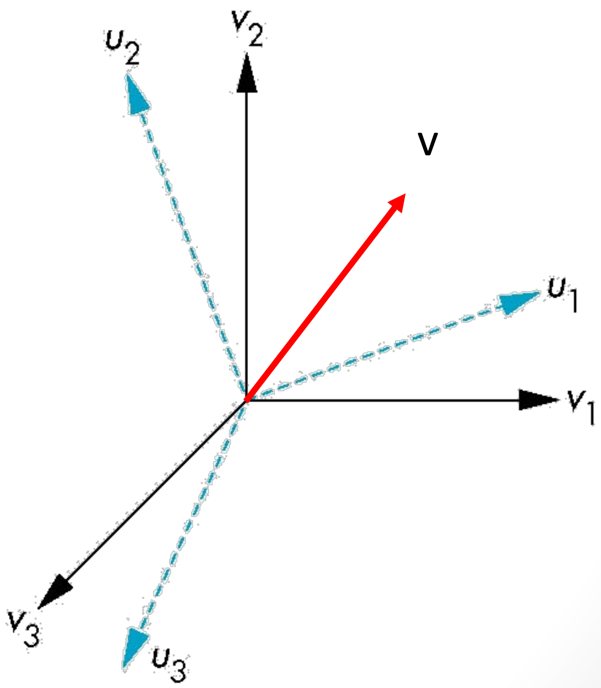
"Change" of Coordinate Systems

- Consider two representations of a the same vector *v* with respect to two different bases. The representations are

$$\mathbf{a} = [\alpha_1\ \alpha_2\ \ \alpha_3]^T$$
$$\mathbf{b} = [\beta_1\ \ \beta_2\ \ \beta_3]^T$$

where

$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = [\alpha_1\ \alpha_2\ \alpha_3]\ [v_1\ v_2\ v_3]^T$$
$$= \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 = [\beta_1\ \beta_2\ \beta_3]\ [u_1\ u_2\ u_3]^T$$



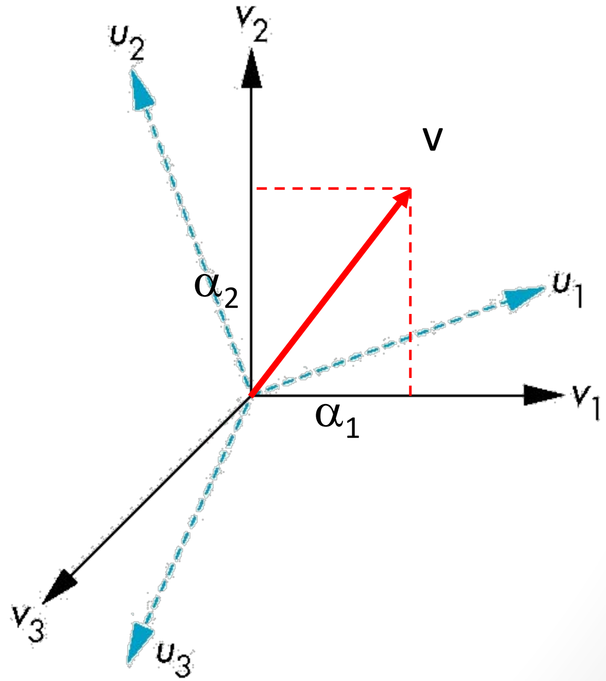
"Change" of Coordinate Systems

- Consider two representations of a the same vector *v* with respect to two different bases. The representations are

$$\mathbf{a} = [\alpha_1\ \alpha_2\ \ \alpha_3]^T$$
$$\mathbf{b} = [\beta_1\ \ \beta_2\ \ \beta_3]^T$$

where

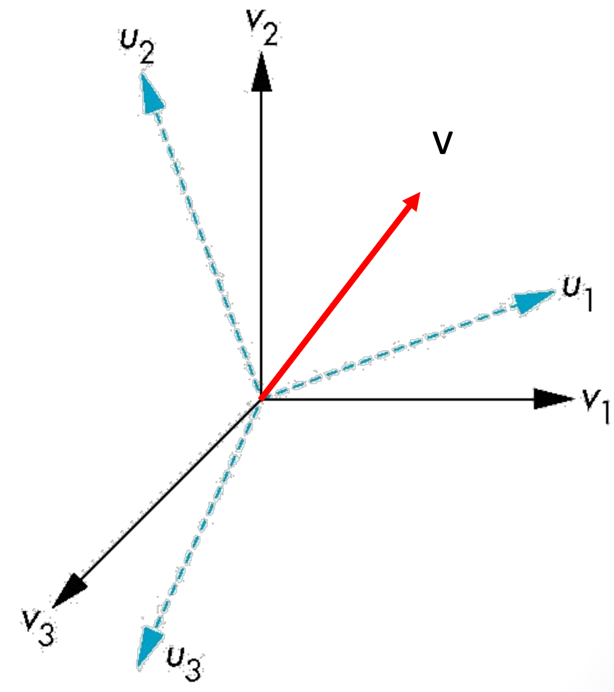
$$v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = [\alpha_1\ \alpha_2\ \alpha_3]\ [v_1\ v_2\ v_3]^T$$
$$= \beta_1 u_1 + \beta_2 u_2 + \beta_3 u_3 = [\beta_1\ \beta_2\ \beta_3]\ [u_1\ u_2\ u_3]^T$$



Representing second basis in terms of first

Each of the basis vectors, u_1,u_2, u_3, are vectors that can be represented in terms of the first basis

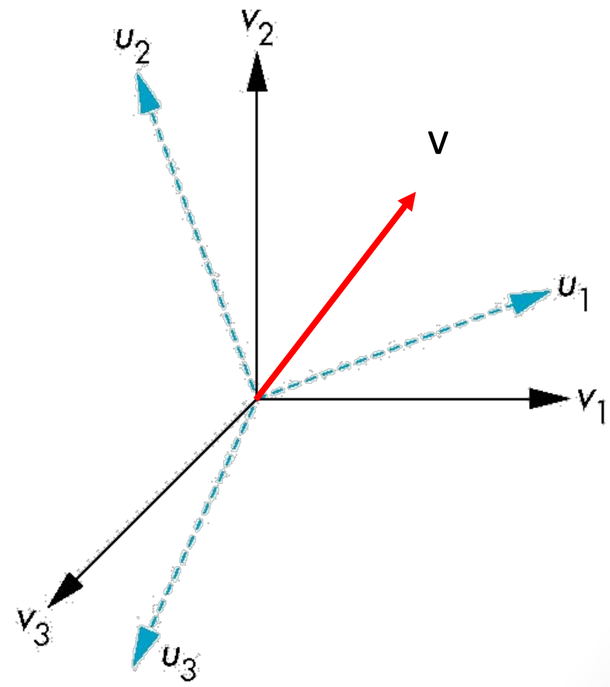
$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$
$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$
$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$



Representing second basis in terms of first

Each of the basis vectors, u_1,u_2, u_3, are vectors that can be represented in terms of the first basis

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$
$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$
$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$



Matrix Form

The coefficients define a 3 x 3 matrix

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

and the *basis* can be related by

$$\boxed{\mathbf{a} = \mathbf{M}^{\mathrm{T}}\mathbf{b}}$$

Example of "Change of representation"

Vector w *whose representation under basis* (v_1, v_2, v_3)

$$\mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$w = v_1 + 2v_2 + 3v_3$

*a new basis

$u_1 = v_1$

$u_2 = v_1 + v_2$

$u_3 = v_1 + v_2 + v_3$

$$\Rightarrow \quad \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Example of "Change of representation" cont.

Matrix

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

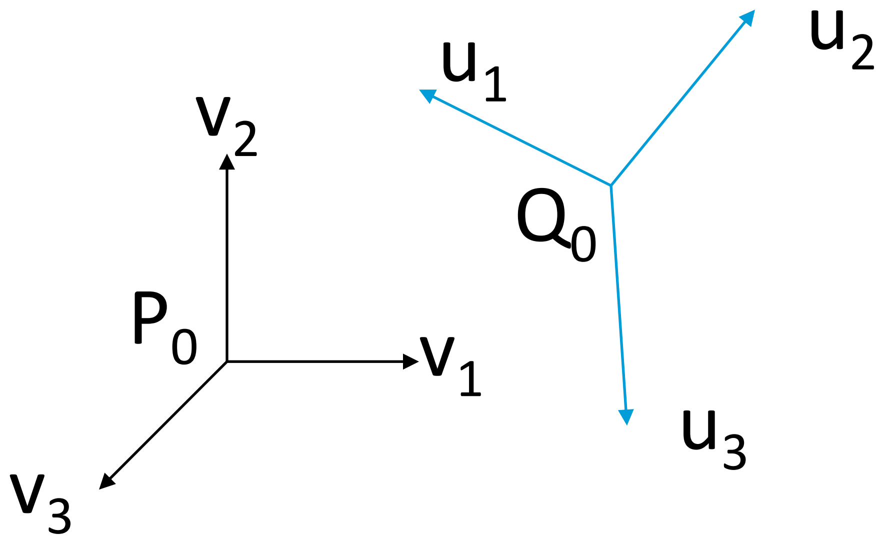
$$\mathbf{A} = \left(\mathbf{M}^{\mathrm{T}}\right)^{-1} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{b} = \mathbf{A}\underline{\mathbf{a}} = \begin{bmatrix} -1 \\ -1 \\ 3 \end{bmatrix} \longleftarrow \mathbf{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

“Change” of Frames

- We can apply a similar process in homogeneous coordinates to the representations of both points and vectors
- Consider two frames

  (P_0, v_1, v_2, v_3)
  (Q_0, u_1, u_2, u_3)



- Any point or vector can be represented in each

Representing One Frame in Terms of the Other

Extending what we did with change of bases

$$
\begin{aligned}
u_1 &= \gamma_{11}v_1+\gamma_{21}v_2+\gamma_{13}v_3 \\
u_2 &= \gamma_{21}v_1+\gamma_{22}v_2+\gamma_{23}v_3 \\
u_3 &= \gamma_{31}v_1+\gamma_{32}v_2+\gamma_{33}v_3 \\
Q_0 &= \gamma_{41}v_1+\gamma_{42}v_2+\gamma_{43}v_3 +P_0
\end{aligned}
$$

defining a 4 x 4 matrix

$$
\mathbf{M} = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}
$$

Working with Representations

Within the two frames any point or vector has a representation of the same form

$\mathbf{a}=[\alpha_1\ \alpha_2\ \alpha_3\ \alpha_4]$ in the first frame
$\mathbf{b}=[\beta_1\ \beta_2\ \beta_3\ \beta_4]$ in the second frame

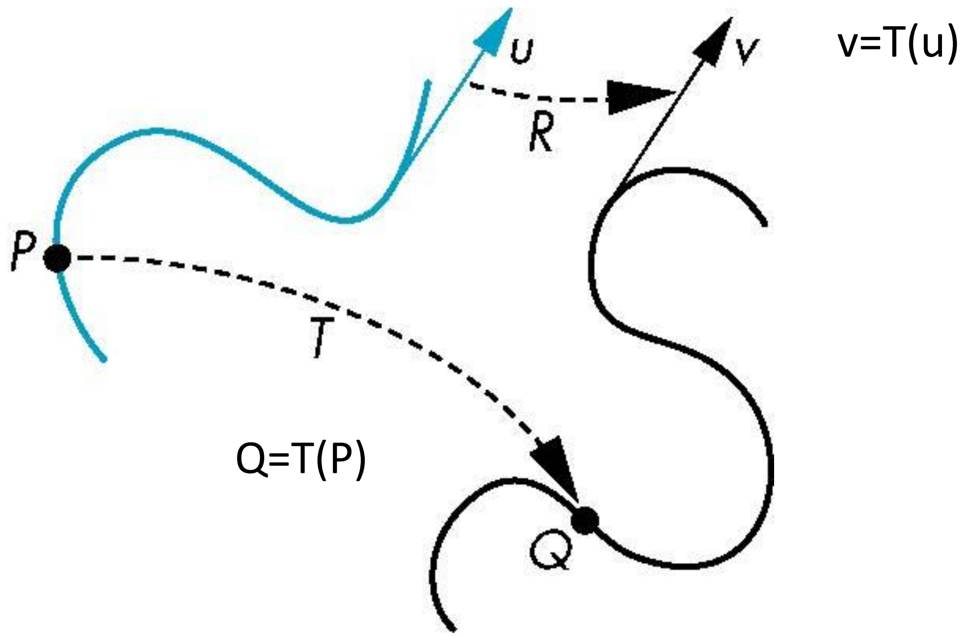
where $\alpha_4 = \beta_4 = 1$ for points and $\alpha_4 = \beta_4 = 0$ for vectors and

$$\mathbf{a} = \mathbf{M}^{\mathrm{T}}\mathbf{b}$$

The matrix $\mathbf{M}$ is 4 x 4 and specifies an affine transformation in homogeneous coordinates

General Transformations

- A transformation maps points to other points and/or vectors to other vectors



Affine Transformations

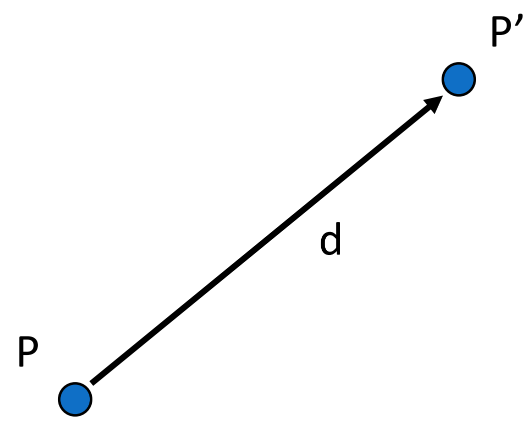
- Every linear transformation is equivalent to a change in frames
- Every affine transformation is a mapping function between affine spaces which preserves points, straight lines and planes.
- However, an affine transformation has only *12 degrees of freedom* because 4 of the elements in the matrix are fixed and are a subset of all possible 4 x 4 linear transformations

Affine Transformations

- Parallel lines remain parallel after an affine transformation
- Characteristic of many physically important transformations
  - Rigid body transformations: rotation, translation
  - Scaling and Shearing.
- The importance (in the graphics) is that…
  - We need only transform endpoints of line segments and let implementation draw line segment between the transformed endpoints

Translation

- Move (translate, displace) a point to a new location



- Displacement determined by a vector d
  - Three degrees of freedom
  - P'=P+d

Translation Using Representations

Using the homogeneous coordinate representation in some frame

$$\mathbf{p} = [x \;\; y \;\; z \;\; 1]^T$$

$$\mathbf{p}' = [x' \; y' \; z' \; 1]^T$$

$$\mathbf{d} = [d_x \; d_y \; d_z \; 0]^T$$

Hence $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ or

$$x' = x + d_x$$

$$y' = y + d_y$$

$$z' = z + d_z$$

note that this expression is in four dimensions and expresses that point = vector + point

Translation Matrix

We can also express translation using a
4 x 4 matrix $\mathbf{T}$ in homogeneous coordinates
$\mathbf{p}' = \mathbf{Tp}$ where

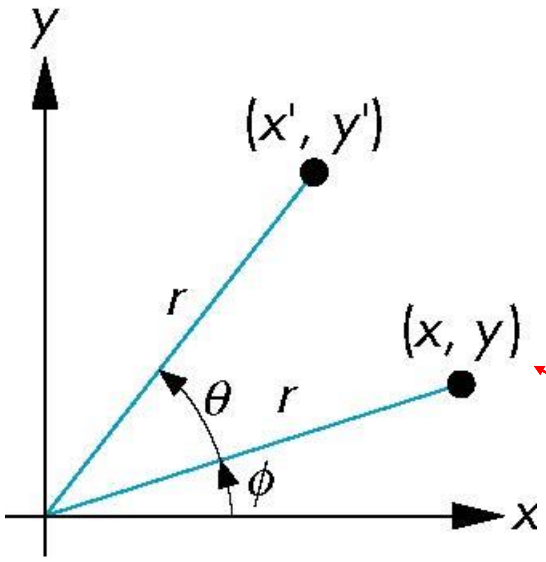
$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together

Rotation (2D)

- Consider rotation about the **origin** by θ degrees
  - radius stays the same, angle increases by θ

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



$$x = r\cos\phi$$
$$y = r\sin\phi$$

$$\boxed{\begin{aligned} x' &= x\cos\theta - y\sin\theta \\ y' &= x\sin\theta + y\cos\theta \end{aligned}}$$

Rotation about the z axis

- Rotation about z axis in three dimensions leaves all points with the same z
  - Equivalent to rotation in two dimensions in planes of constant z

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

  - or in homogeneous coordinates

$$\mathbf{p}' = \mathbf{R}_{\mathbf{z}}(\theta)\mathbf{p}$$

Rotation Matrix

$$\mathbf{R} = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & \boxed{1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about x and y axes

- Same argument as for rotation about *z* axis
  - For rotation about x axis, x is unchanged
  - For rotation about y axis, y is unchanged

$$\mathbf{R} = \mathbf{R}_x(\theta) = \begin{bmatrix} \boxed{1} & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{R} = \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & \boxed{1} & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

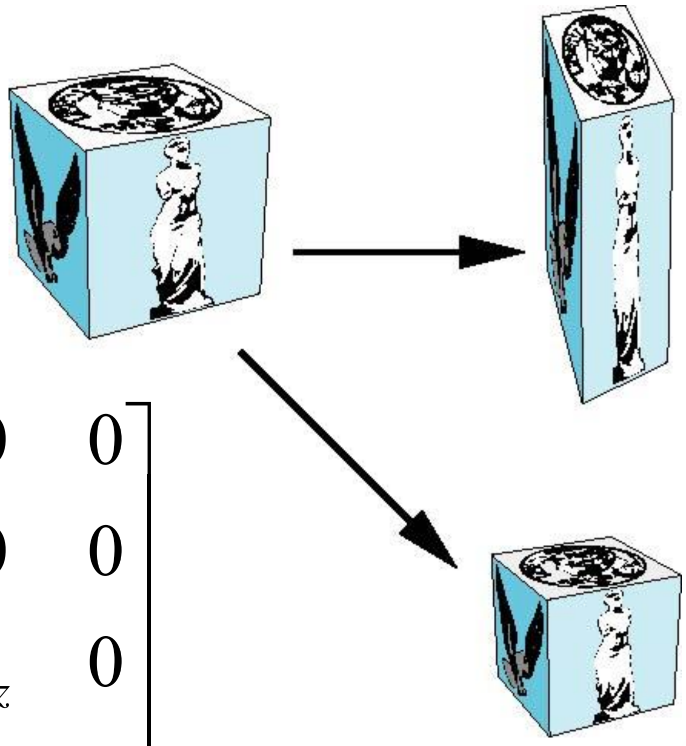
Scaling

Expand or contract along each axis (fixed point of origin)

$x' = s_x x$
$y' = s_y y$
$z' = s_z z$

p'= S p

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

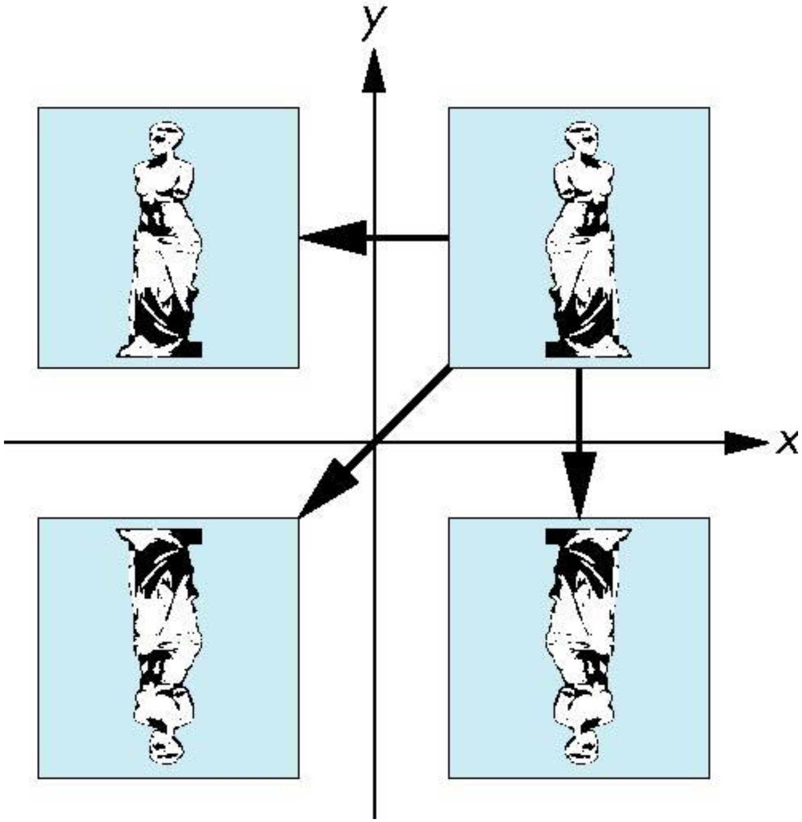


Reflection

corresponds to negative scale factors

$s_x = -1$ $s_y = 1$

original



$s_x = -1$ $s_y = -1$

$s_x = 1$ $s_y = -1$

Inverses

- Although we could compute inverse matrices by general formulas, we can use simple geometric observations
  - Translation: $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
  - Rotation: $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$
    - Holds for any rotation matrix
    - Note that since $\cos(-\theta) = \cos(\theta)$ & $\sin(-\theta) = -\sin(\theta)$ **>>** $\mathbf{R}^{-1}(\theta) = \mathbf{R}^{T}(\theta)$
  - Scaling: $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

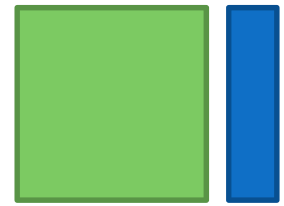
Concatenation

- We can form arbitrary affine transformation matrices by multiplying them together.
  - Ex: rotation, translation, and scaling matrices…
- Because the same transformation is applied to many vertices, it is more efficient to pre-compute a matrix $\mathbf{M}=\mathbf{ABC}$ and compute $\mathbf{Mp}$ for many vertices $\mathbf{p}$
- The difficult part is how to form a desired transformation from the specifications in the application

Order of Transformations

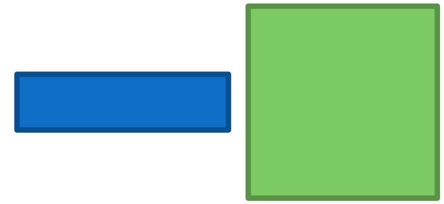
- Note that matrix on the right is the first applied
- Mathematically, the following are equivalent

$$\mathbf{p' = ABCp = A(B(Cp))}$$

Cp = 

- Note many references use *row major matrices* to present points.

$$\mathbf{p}^{T'} = \mathbf{p}^{T}\mathbf{C}^{T}\mathbf{B}^{T}\mathbf{A}^{T}$$



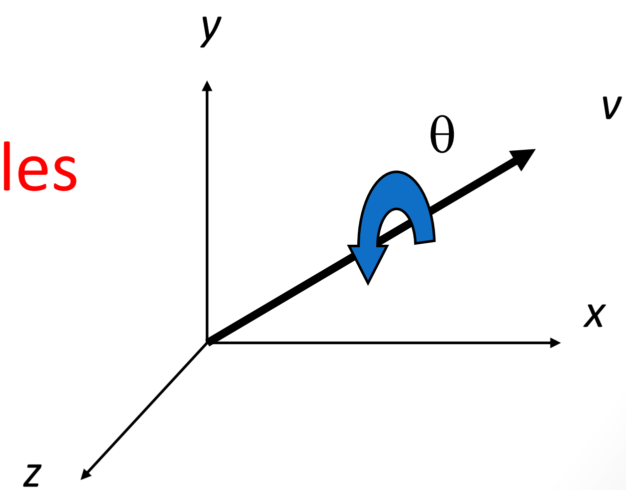
General Rotation About the Origin

A rotation by θ about an arbitrary axis can be decomposed into the concatenation of rotations about the *x*, *y*, and *z* axes

$$\mathbf{R}(\theta) = \mathbf{R}_1(\theta_1)\,\mathbf{R}_2(\theta_2)\,\mathbf{R}_3(\theta_3)$$

$\theta_1\ \theta_2\ \theta_3$ are called the Euler angles

We can use rotations in another order but with different angles



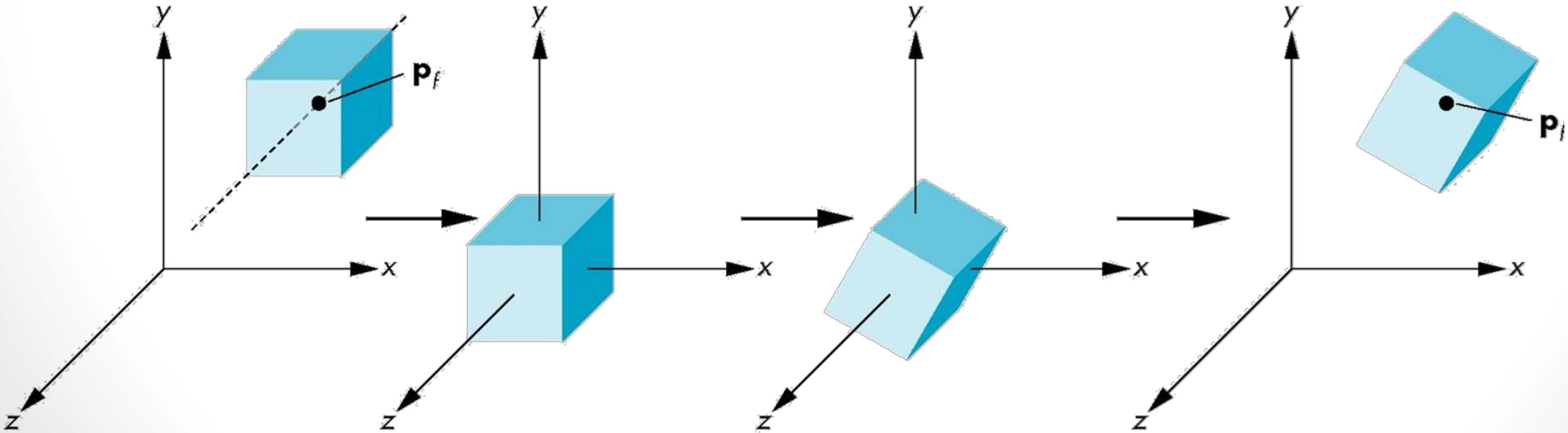
Rotation About a Fixed Point other than the Origin

Move fixed point to origin

Rotate

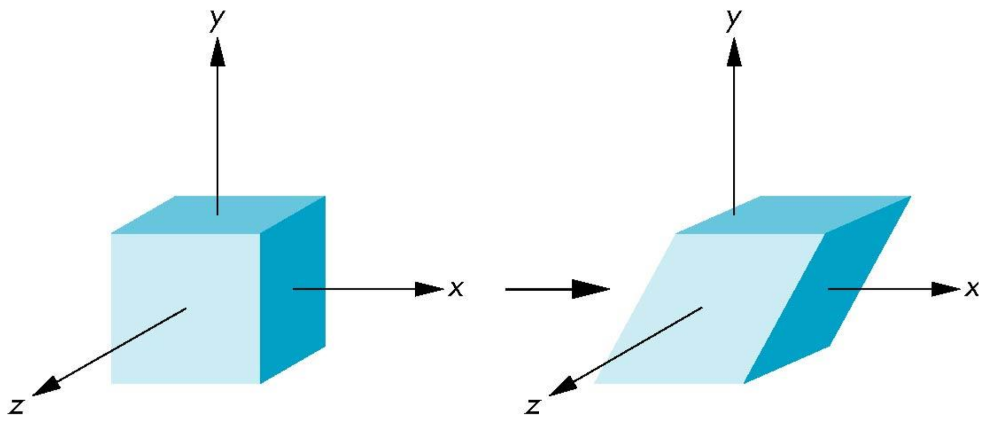
Move fixed point back

$\mathbf{M} = \mathbf{T}(p_f)\, \mathbf{R}(\theta)\, \mathbf{T}(-p_f)$



Shear

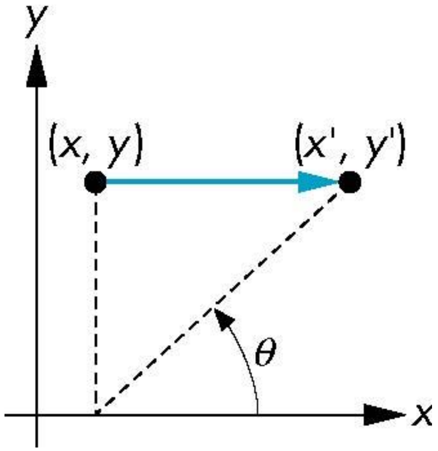
- Helpful to add one more basic transformation
- Equivalent to pulling faces in opposite directions



Shear Matrix

Consider simple shear along x axis

$x' = x + y \cot \theta$
$y' = y$
$z' = z$

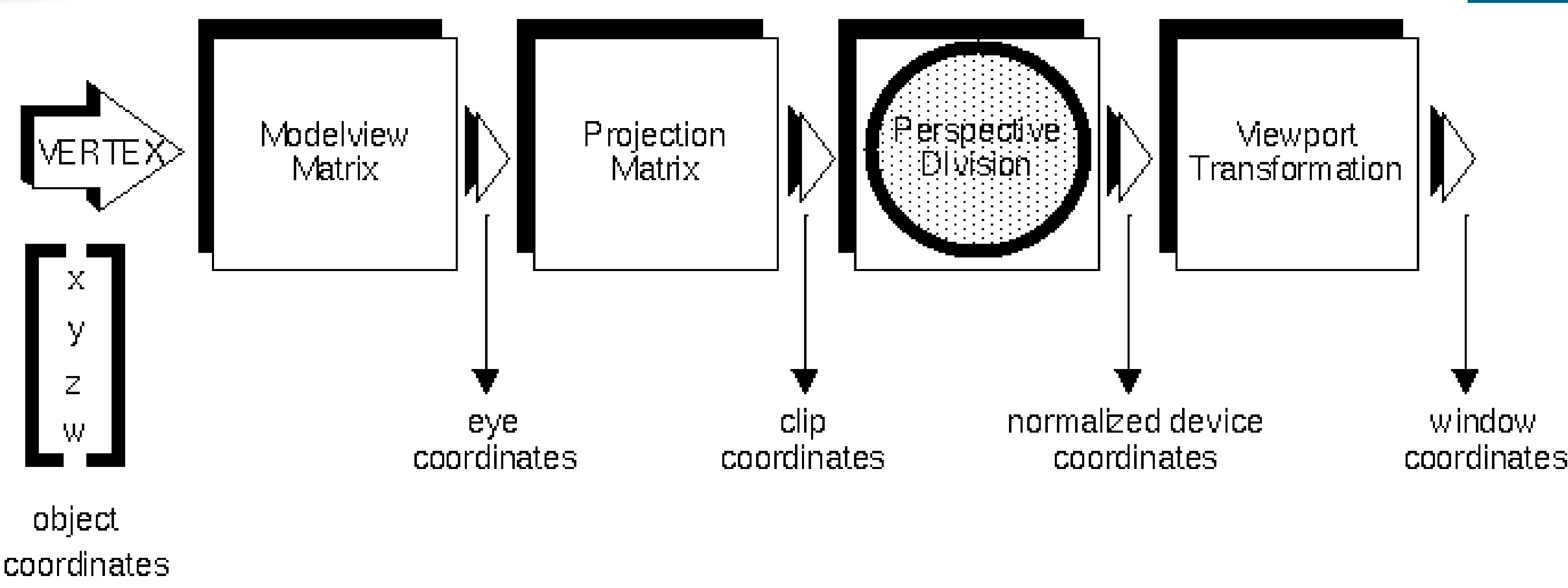


$$\mathbf{H}(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OpenGL Matrices

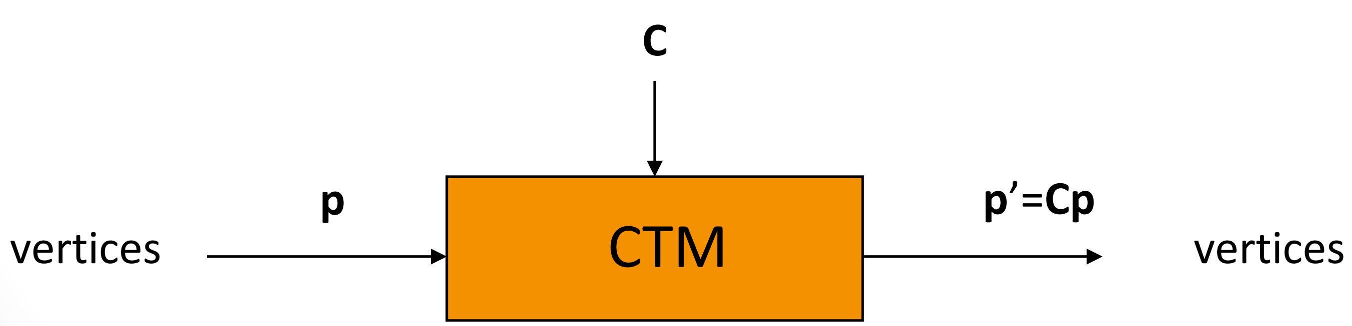
- In OpenGL matrices are part of the state
- Three types
  - Model-View (**`GL_MODEL_VIEW`**)
  - Projection (**`GL_PROJECTION`**)
  - Texture (**`GL_TEXTURE`**) (ignore for now)
- Single set of functions for manipulation
- Select which to manipulated by
  - **`glMatrixMode(GL_MODEL_VIEW);`**
  - **`glMatrixMode(GL_PROJECTION);`**

Stages of Vertex Transformation



Current Transformation Matrix

- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the *current transformation matrix* (CTM) that is part of the state and is applied to all vertices that pass down the pipeline
- The CTM is defined in the user program and loaded into a transformation unit



CTM operations

- The CTM can be altered either by loading a new CTM or by postmutiplication

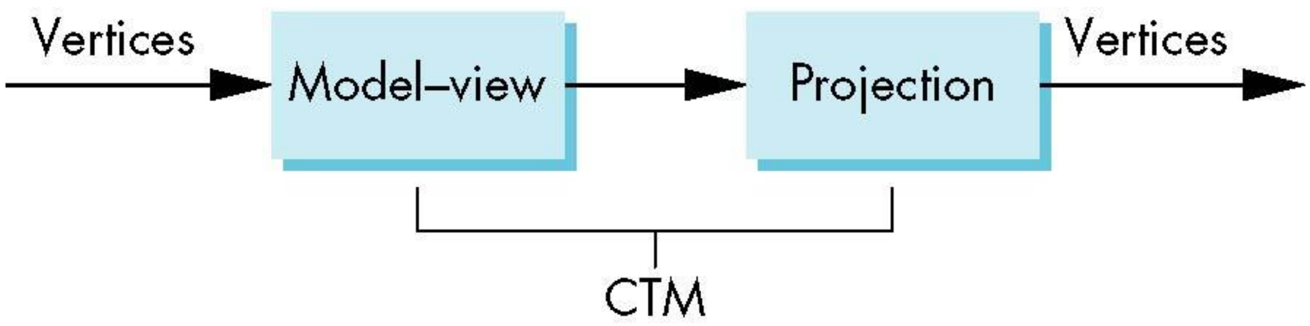
Load an identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Load an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{M}$

Load a translation matrix: $\mathbf{C} \leftarrow \mathbf{T}$
Load a rotation matrix: $\mathbf{C} \leftarrow \mathbf{R}$
Load a scaling matrix: $\mathbf{C} \leftarrow \mathbf{S}$

Post-multiply by an arbitrary matrix: $\mathbf{C} \leftarrow \mathbf{CM}$
Post-multiply by a translation matrix: $\mathbf{C} \leftarrow \mathbf{CT}$
Post-multiply by a rotation matrix: $\mathbf{C} \leftarrow \mathbf{C\,R}$
Post-multiply by a scaling matrix: $\mathbf{C} \leftarrow \mathbf{C\,S}$

Rotation, Translation, Scaling CTM in OpenGL

- OpenGL has a model-view and a projection matrix in the pipeline which are concatenated together to form the CTM
- Can manipulate each after setting the matrix mode: glMatrixMode(MATRIX_NAME)



Rotation, Translation, Scaling

Load an identity matrix:

```
glLoadIdentity()
```

Multiply on right:

```
glRotatef(theta, vx, vy, vz)
```

theta is in degrees, (**vx, vy, vz**) represent the axis of rotation

```
glTranslatef(dx, dy, dz)

glScalef(sx, sy, sz)
```

Each axis has a float (f) value. We use double (d) format in **glScaled**

Arbitrary Matrices

- Can load and multiply by matrices defined in the application program

```
glLoadMatrixf(m)
glMultMatrixf(m)
```

- The matrix ***m*** is a one dimension array of 16 elements which are the components of the desired 4 x 4 matrix stored by columns
- In **`glMultMatrixf`**, **`m`** multiplies the existing matrix on the right

Rotation about a Fixed Point

Start with identity matrix: $\mathbf{C} \leftarrow \mathbf{I}$
Move fixed point back: $\mathbf{C} \leftarrow \mathbf{CT}$
Rotate: $\mathbf{C} \leftarrow \mathbf{CR}$
Move fixed point to origin: $\mathbf{C} \leftarrow \mathbf{CT}^{-1}$

Result: $\mathbf{C} = \mathbf{TRT}^{-1}$

Each operation corresponds to one function call in the program.

Note that the last operation specified is the first executed in the program

Example

- Perform a 45 degrees rotation about the line through the origin and the point (1.0,2.0,3.0) with a fixed point of (4.0, 5.0, 6.0)

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(4.0, 5.0, 6.0);
glRotatef(45.0, 1.0, 2.0, 3.0);
glTranslatef(-4.0, -5.0, -6.0);
```

- Remember that last matrix specified in the program is the first applied

Order of transformations

- The last matrix specified in the program is the first applied
  - **$C \leftarrow I$**
  - **$C \leftarrow CT(4.0, 5.0, 6.0)$**
  - **$C \leftarrow CR(45.0, 1.0, 2.0, 3.0)$**
  - **$C \leftarrow CT(-4.0, -5.0, -6.0)$**
- **Multiply at the end of CTM**
  - **$C \leftarrow CT(4.0, 5.0, 6.0)\ R(45.0, 1.0, 2.0, 3.0)\ T(-4.0, -5.0, -6.0)$**

Reading Back Matrices

- Can also access matrices (and other parts of the state) by *enquiry (query)* functions

```
glGetIntegerv
glGetFloatv
glGetBooleanv
glGetDoublev
glIsEnabled
```

- For matrices, we use as

```
float m[16];
glGetFloatv(GL_MODELVIEW, m);
```

Using the Model-View Matrix

- In OpenGL the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but is often easier to use **`gluLookAt`** (Chapter 5)
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens

Using the Model-View Matrix

- Although both are manipulated by the same functions, we have to be careful because incremental changes are always made by post-multiplication
  - For example, rotating model-view and projection matrices by the same matrix are not equivalent operations.
  - Post-multiplication of the model-view matrix is equivalent to pre-multiplication of the projection matrix

Matrix Stacks

- In many situations we want to save transformation matrices (current state) for use later
  - Traversing hierarchical data structures (Chapter 9)
  - Avoiding state changes wlhen executing display lists
- OpenGL maintains stacks for each type of matrix
  - Access present type (as set by **`glMatrixMode)`** by

```
glPushMatrix()
glPopMatrix()
```

Loading, pushing, and poping matrices

Perform a transformation and then return to the same state as before its execution

```
glPushMatrix();
glTranslatef(…);
glRotatef(…);
glScalef(…);
/* draw object here */
glPopMatrix();
```

Check this one for more detail…

- OpenGL Programming - Push and Pop Matrix.pptx

SUGGESTION!

OR

OBJECTION?

Let's stop here,

TAKE A BREAK