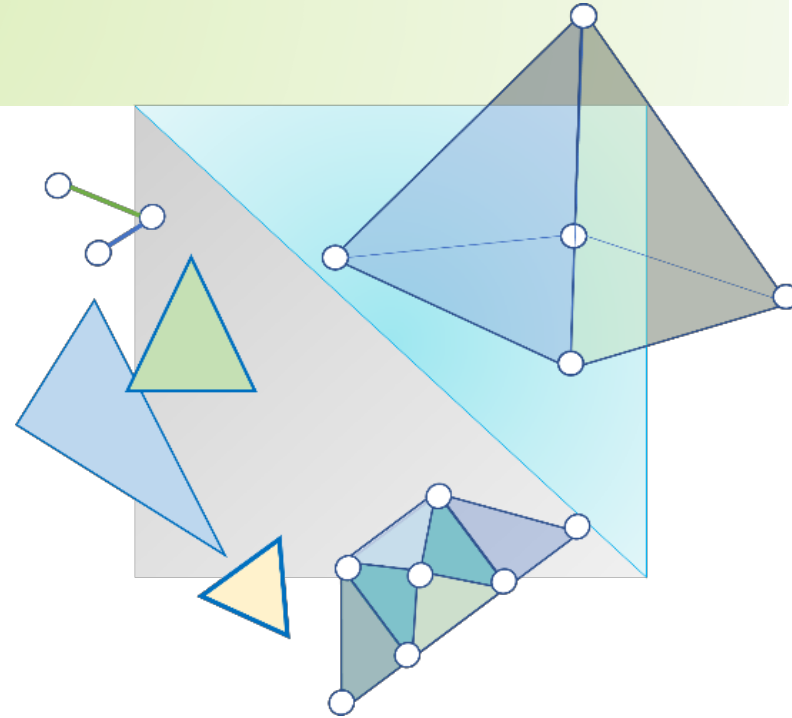


CITS3003 Graphics & Animation

Lecture 8: Coordinate Frames and Transformations



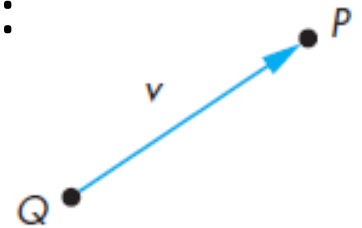
Content

- Introduce standard transformations
 - Rotation, Translation, Scaling, Shear
- Learn to build arbitrary transformation matrices from simple transformations

Revision

- Vectors are entities having length and direction, but no position.
- The **difference** between two points is a vector:

$$\mathbf{v} = \mathbf{P} - \mathbf{Q}$$



- The **sum** of a point and a vector is a point:

$$\mathbf{P} = \mathbf{v} + \mathbf{Q}$$

we also say that a point \mathbf{P} is formed by displacing point \mathbf{Q} by vector \mathbf{v}

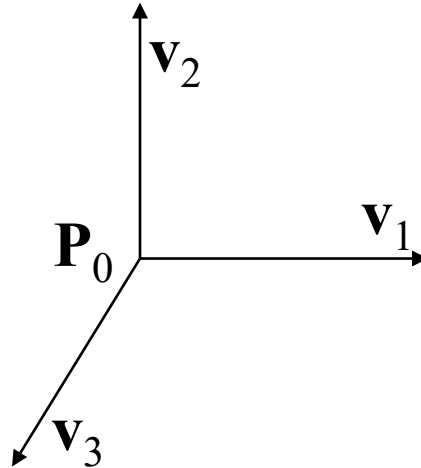
- An n-dimensional vector is given by an n-tuple (list of its components)

$$\mathbf{w} = [\alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_n]^T$$

representations of \mathbf{w} with respect to the basis vectors

A Coordinate Frame

- A coordinate system (or coordinate frame) is determined by $(\mathbf{P}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$



The Homogeneous representation of a Point and a Vector

- Assuming $\mathbf{0} \cdot \mathbf{P} = \mathbf{0}$ and $\mathbf{1} \cdot \mathbf{P} = \mathbf{P}$, we can write

$$\mathbf{w} = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 = \alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \alpha_3 \mathbf{v}_3 + \mathbf{0} \cdot \mathbf{P}_0$$

$$\mathbf{P} = \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 + \mathbf{P}_0 = \beta_1 \mathbf{v}_1 + \beta_2 \mathbf{v}_2 + \beta_3 \mathbf{v}_3 + \mathbf{1} \cdot \mathbf{P}_0$$

- Thus, we obtain the four-dimensional *homogeneous coordinate* representation

$$\mathbf{w} = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 0]^T$$

$$\mathbf{P} = [\beta_1 \quad \beta_2 \quad \beta_3 \quad 1]^T$$

The Homogeneous representation of a Point and a Vector

- Homogeneous coordinates help keep the distinction between points and vectors.
- To go from ordinary to homogeneous coordinates:
 - if it's a point append a 1;
 - if it's a vector, append a 0;

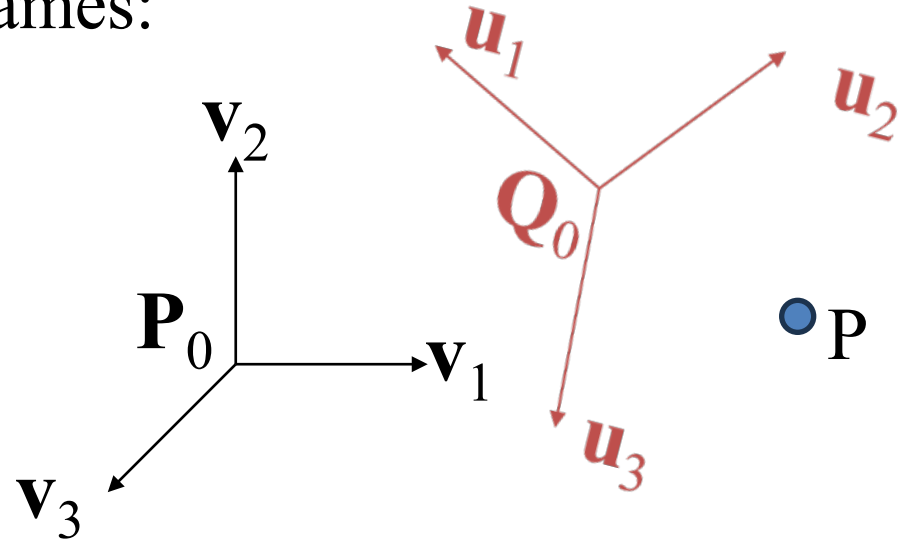
$$\mathbf{w} = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad 0]^T$$

$$\mathbf{P} = [\beta_1 \quad \beta_2 \quad \beta_3 \quad 1]^T$$

Change of Coordinate Frames

Consider two coordinate frames:

$$\begin{aligned} &(\mathbf{P}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3) \\ &(\mathbf{Q}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3) \end{aligned}$$



- We can represent $(\mathbf{Q}_0, \mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3)$ in terms of $(\mathbf{P}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3)$

$$\mathbf{u}_1 = \gamma_{11}\mathbf{v}_1 + \gamma_{12}\mathbf{v}_2 + \gamma_{13}\mathbf{v}_3$$

$$\mathbf{u}_2 = \gamma_{21}\mathbf{v}_1 + \gamma_{22}\mathbf{v}_2 + \gamma_{23}\mathbf{v}_3$$

$$\mathbf{u}_3 = \gamma_{31}\mathbf{v}_1 + \gamma_{32}\mathbf{v}_2 + \gamma_{33}\mathbf{v}_3$$

$$\mathbf{Q}_0 = \gamma_{41}\mathbf{v}_1 + \gamma_{42}\mathbf{v}_2 + \gamma_{43}\mathbf{v}_3 + \mathbf{P}_0$$

Representing One Coordinate Frame in Terms of the Other

$$\mathbf{u} = \mathbf{M}\mathbf{v}$$

\mathbf{M} is a 4×4 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 coordinate frames any point or vector can be represented as:

$\mathbf{a} = [\alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4]$ in the first frame

$\mathbf{b} = [\beta_1 \quad \beta_2 \quad \beta_3 \quad \beta_4]$ in the second frame

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

$$\mathbf{a} = \mathbf{M}^T \mathbf{b} \quad \text{or}$$

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

where,
 $\mathbf{T} = (\mathbf{M}^T)^{-1}$

- The matrix \mathbf{M}^T is 4×4 and specifies an **affine transformation** in homogeneous coordinates

An Example

We consider two reference frames that have basis vector relation

$$u_1 = v_1,$$

$$u_2 = v_1 + v_2,$$

$$u_3 = v_1 + v_2 + v_3.$$

Let's say the reference point does not change, so

$$Q_0 = P_0.$$

Our matrix M^T would be:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow M^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \longleftarrow \text{Only accounting for rotation}$$

An Example

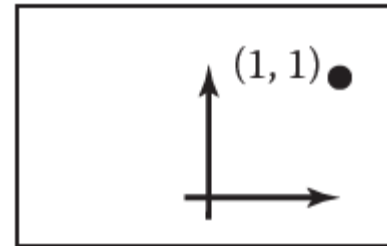
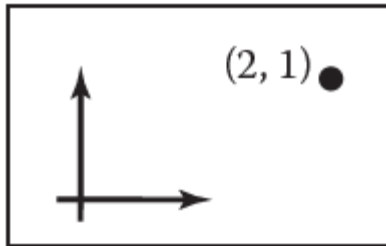
Now, we want our frames to have different reference point....
Let's say, to the point Q_0 that has the following representation in the original system.

$$Q_0 = P_0 + v_1 + 2v_2 + 3v_3,$$

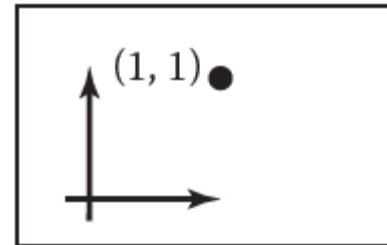
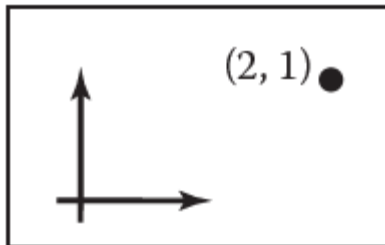
The M^T for such a setting will be:

$$M^T = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \leftarrow \text{Also accounting for translation}$$

Transformations



changing the
coordinate
System
(Lec#07)



using
transformation
matrices
to move points
around
(today's lecture)

On the top right is our mental image if we view it as a change of coordinates (a movement of the origin in this case).
and on the bottom right is our mental image if we view this transformation as a physical movement
Both ways will lead to exactly the same matrix

Matrix Multiplication

$$\begin{bmatrix} 1 & 1.1 \\ 2 & 2.2 \\ 3 & 3.3 \end{bmatrix} \times \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1a + 1.1b \\ 2a + 2.2b \\ 3a + 3.3b \end{bmatrix}$$
$$= a \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + b \begin{bmatrix} 1.1 \\ 2.2 \\ 3.3 \end{bmatrix}$$

$a = b = 1$

$$= \begin{bmatrix} 2.1 \\ 4.2 \\ 6.3 \end{bmatrix}$$

Identity Matrix

The *identity matrix* contains all zeros, with ones along the diagonal

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

Any point or matrix multiplied by the identity matrix is unchanged.

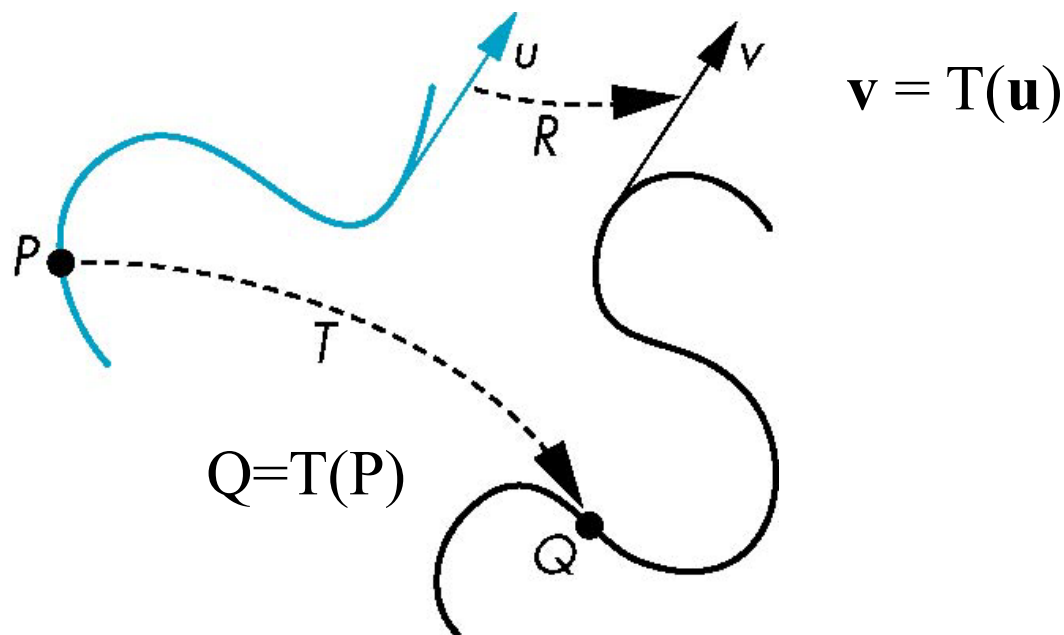
Transpose of a Matrix

The *transpose* of a matrix is computed by interchanging its rows and columns.
For example:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{10} & A_{20} & A_{30} \\ A_{01} & A_{11} & A_{21} & A_{31} \\ A_{02} & A_{12} & A_{22} & A_{32} \\ A_{03} & A_{13} & A_{23} & A_{33} \end{bmatrix}^T$$

Transformation

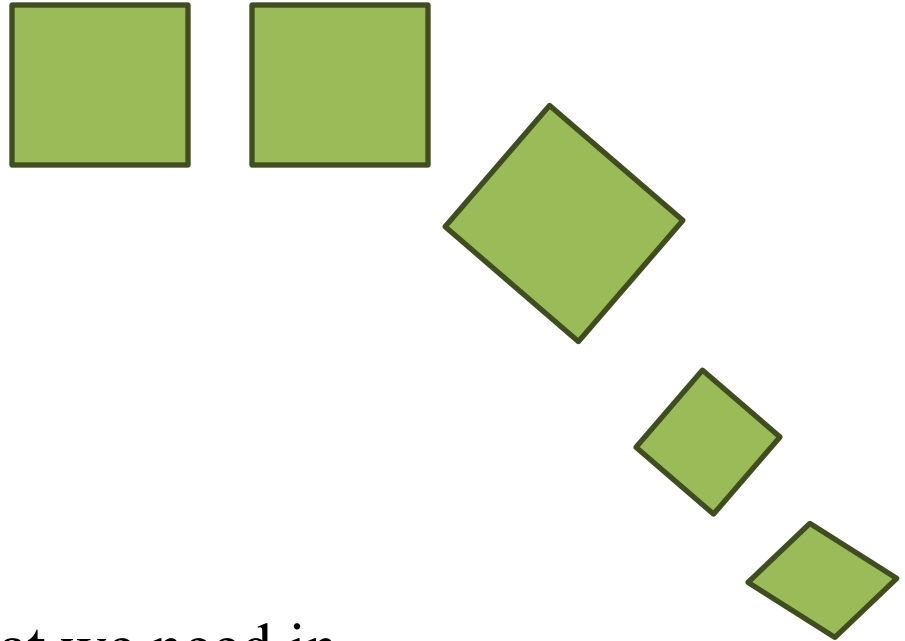
- A transformation is a function that maps points to other points and/or vectors to other vectors



Transformation matrices

In graphics, matrices are typically used for performing transformations on objects. For example, a matrix can be used to move a point from one location to another. We will learn several useful transformation matrices:

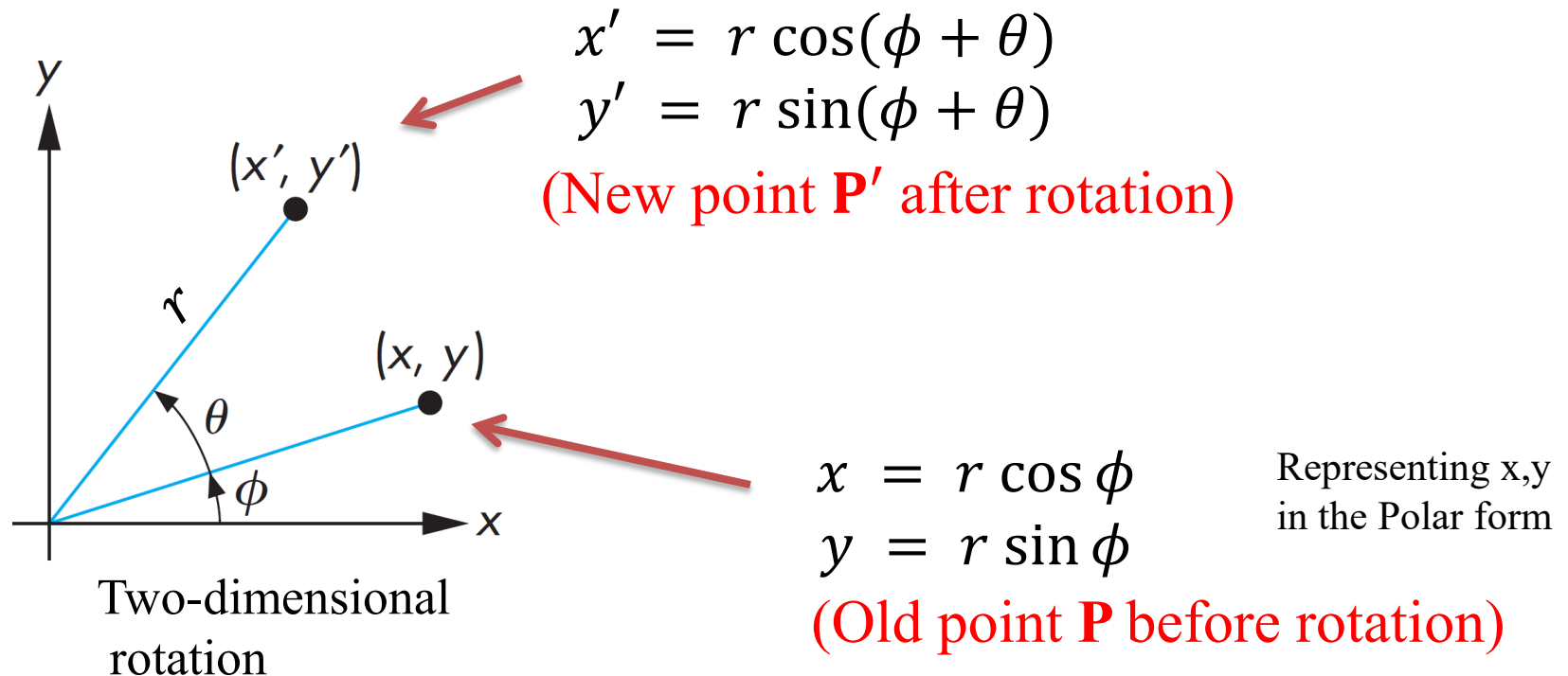
- Translation
- Rotation
- Scale
- Shear



Most of the transformations that we need in computer graphics are affine.

Rotation (2D)

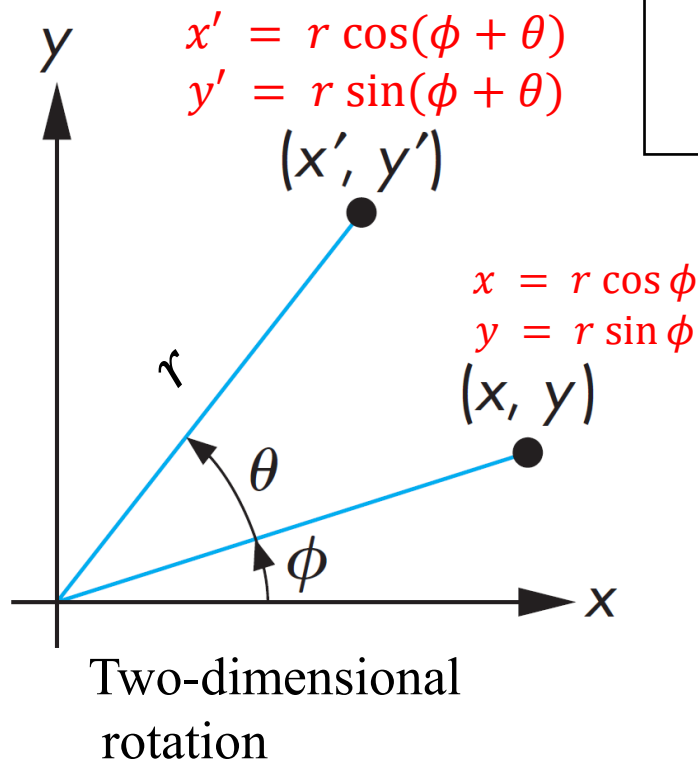
- Consider a rotation about the origin by θ degrees
 - radius stays the same, angle increases by θ



$$\begin{aligned}\sin(A+B) &= \sin(A) \cos(B) + \cos(A) \sin(B) \\ \cos(A+B) &= \cos(A) \cos(B) - \sin(A) \sin(B)\end{aligned}$$

Rotation (2D)

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



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

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

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

$$\sin(A+B) = \sin(A) \cos(B) + \cos(A) \sin(B)$$

$$\cos(A+B) = \cos(A) \cos(B) - \sin(A) \sin(B)$$

Rotation (2D)

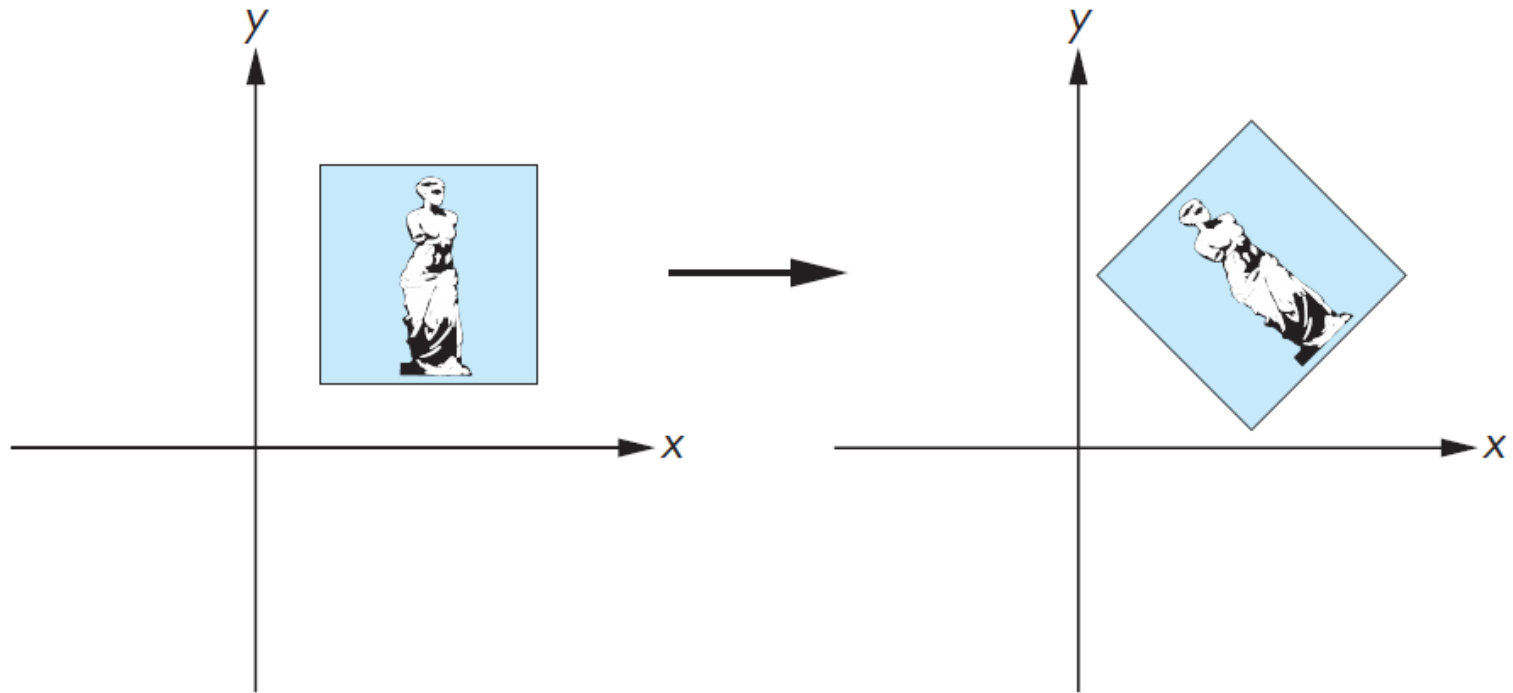


FIGURE 3.35 Rotation about a fixed point.

Rotation about the z-axis

- Rotation in two dimensions is equivalent to rotation about the z axis in three dimensions.
- Rotation about the z axis in 3D leaves the z components of all the points unchanged:

$$R_z(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about the z-axis

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

$$\begin{aligned} x' &= x \cos \theta - y \sin \theta + 0 + 0 & = x \cos \theta - y \sin \theta \\ y' &= x \sin \theta + y \cos \theta + 0 + 0 & = x \sin \theta + y \cos \theta \\ z' &= 0 + 0 + 0 + 0 & = 0 \\ w' &= 0 + 0 + 0 + 1 & = 1 \end{aligned}$$

In homogeneous coordinates

$$\mathbf{P}' = \mathbf{R}_z(\theta)\mathbf{P}$$

Rotation about x and y axis

- We can also derive the matrices for rotation about the x and y axis.
 - For rotation about x axis, x values are unchanged
 - For rotation about y axis, y values are unchanged

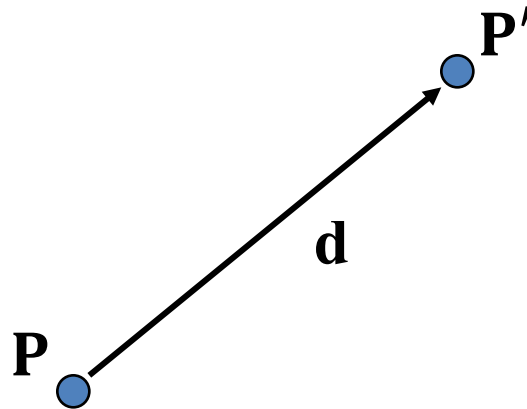
$$\mathbf{R} = \mathbf{R}_x(q) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

Note the negative
sign here

Translation

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

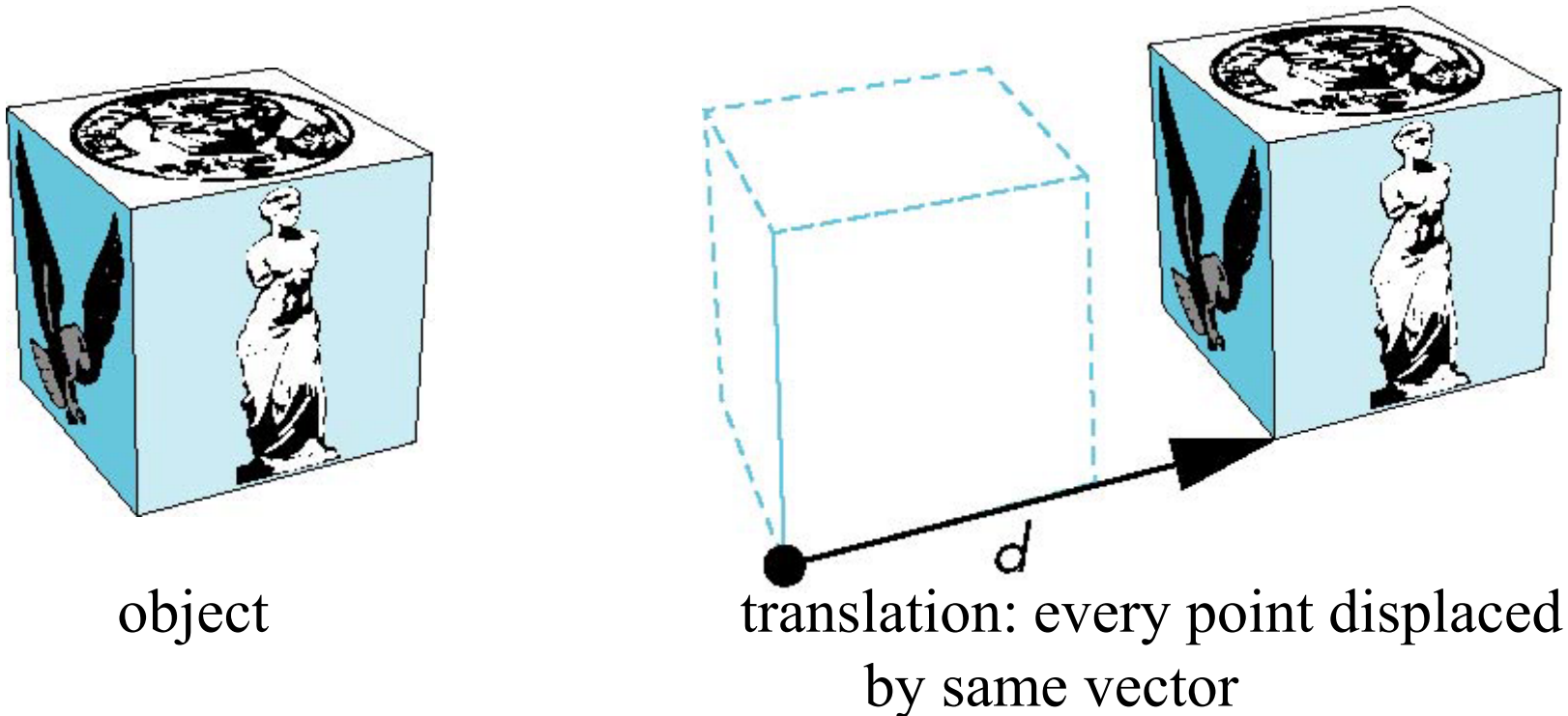


- Displacement is determined by a vector **d**
 - Three degrees of freedom

$$\mathbf{P'} = \mathbf{P} + \mathbf{d}$$


The whole object moves

- Although we can move a point to a new location in an infinite number of ways, when we move many points of a rigid object there is usually only one way



Translation using Representations

- Using the homogeneous coordinate representation in a frame
 - $\mathbf{P} = [x \quad y \quad z \quad 1]^T$
 - $\mathbf{P}' = [x' \quad y' \quad z' \quad 1]^T$
 - $\mathbf{d} = [d_x \quad d_y \quad d_z \quad 0]^T$
- 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 point = point + vector

Translation Matrix

- We can also express translation using a 4×4 matrix \mathbf{T} in homogeneous coordinates

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

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

Translation Matrix

- We can also express translation using a 4×4 matrix **T** in homogeneous coordinates

P' = **TP** where

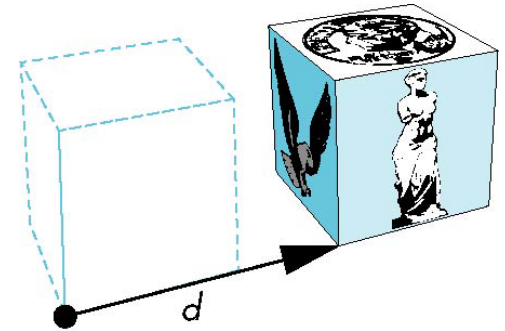
$$\mathbf{P}' = \mathbf{TP} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + d_x \\ y + d_y \\ z + d_z \\ 1 \end{bmatrix}$$

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

Translation

- Translate object= Move each vertex by the same distance $d = (d_x, d_y, d_z)$

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



If we translate a point (2,2,2) by displacement (2,4,6), new location of point is (4,6,8)

Using matrix multiplication for translation

$$\begin{pmatrix} 4 \\ 6 \\ 8 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 6 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \end{pmatrix}$$

Translated
point

Translation Matrix

Original point

Where

- $x' = x + d_x$
- $y' = y + d_y$
- $z' = z + d_z$

Scaling (non-rigid transform)

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

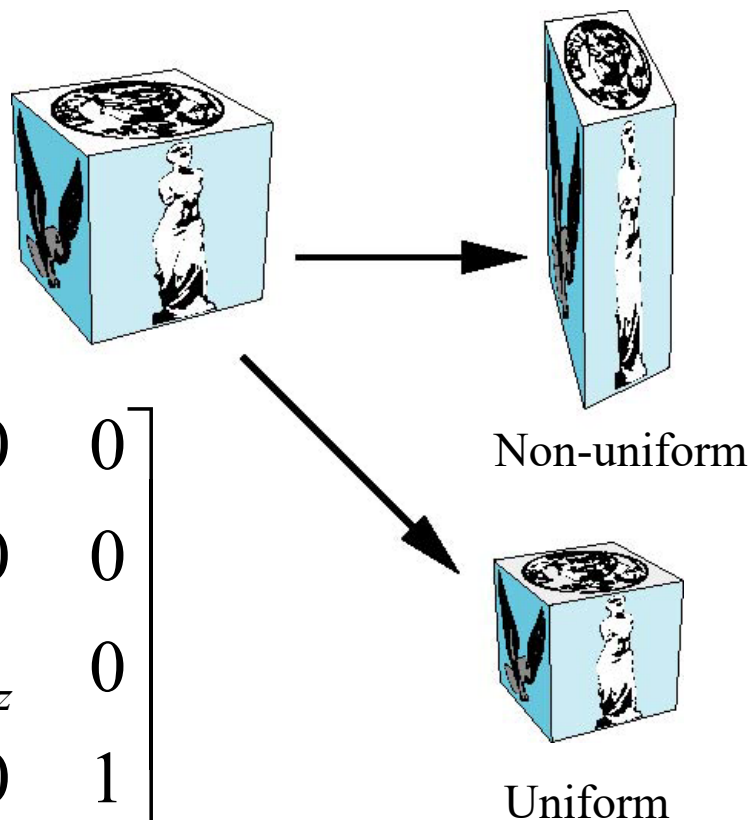
$$x' = s_x x$$

$$y' = s_y y$$

$$z' = s_z z$$

$$\mathbf{P}' = \mathbf{S}\mathbf{P}$$

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

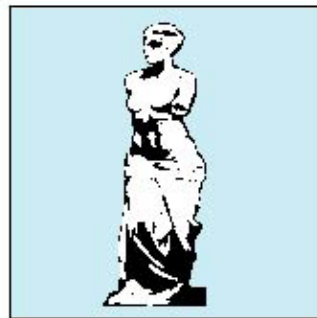


Reflection

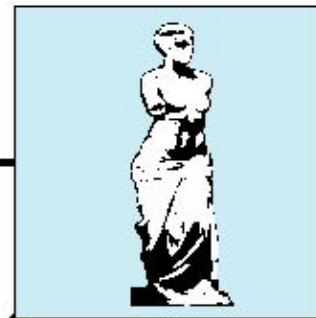
corresponds to negative scale factors

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

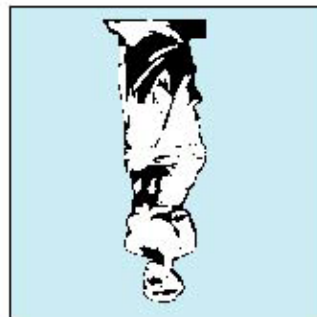
$$\begin{aligned} s_x &= -1 \\ s_y &= 1 \end{aligned}$$



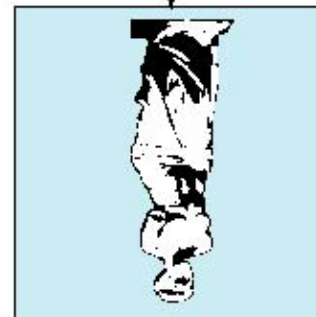
original



$$\begin{aligned} s_x &= -1 \\ s_y &= -1 \end{aligned}$$

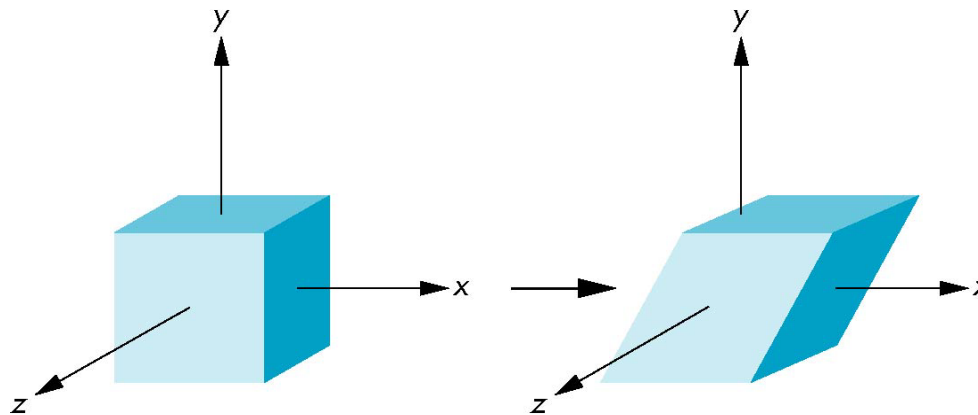


$$\begin{aligned} s_x &= 1 \\ s_y &= -1 \end{aligned}$$



Shear

- It is helpful to add one more basic transformation, the *shearing transformation*, to the collection of transformation we have learnt
- Shearing is equivalent to pulling faces in opposite directions

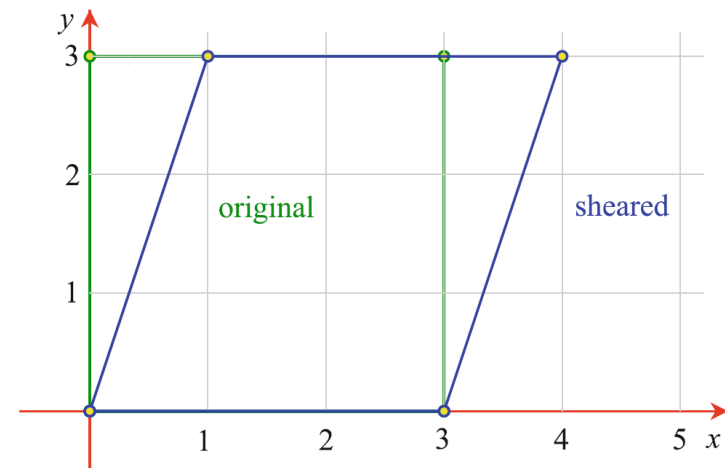
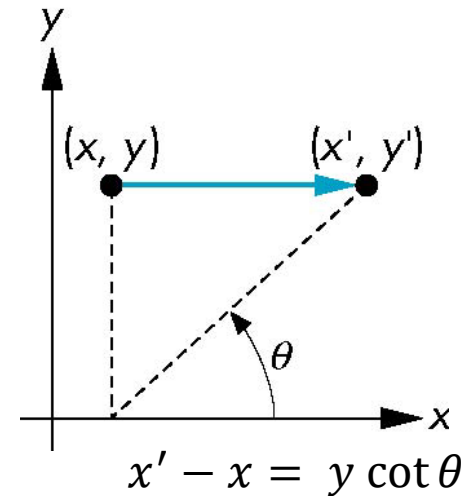


Shear

Consider a simple shear along the ***x*** axis

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

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

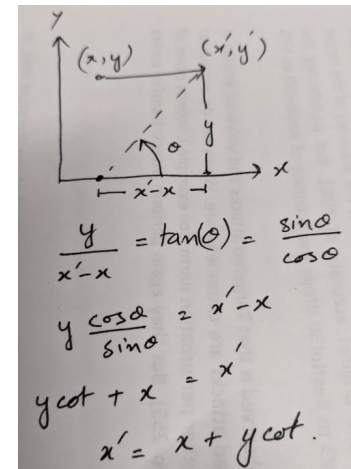
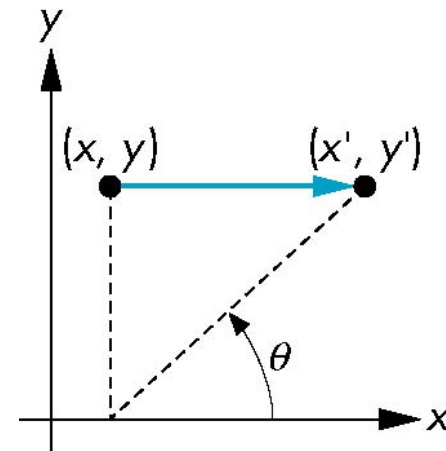


Shear

Consider a simple shear along the x axis

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

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



Inverses

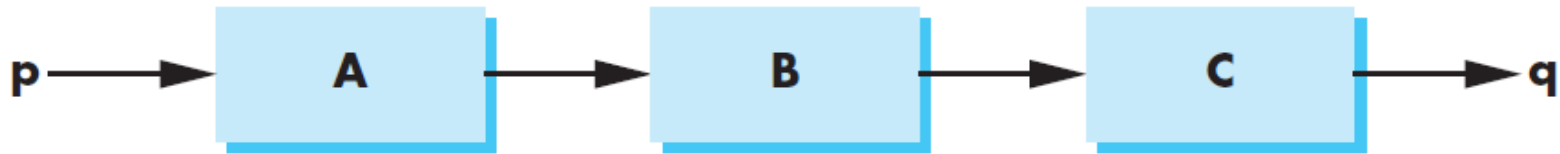
- Although we could compute inverse matrices by general formulas, we can use simple geometric observations
 - **Translation:** $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
 - **Rotation:** $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$
 - Holds for any rotation matrix
 - Note that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

Concatenation

- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, scaling and shear matrices
- Because the same transformation is applied to many vertices, the cost of forming a matrix $\mathbf{M} = \mathbf{ABCD}$ is not significant compared to the cost of computing \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

- Suppose that we carry out three successive transformations on a point **p**, creating a new point **q**

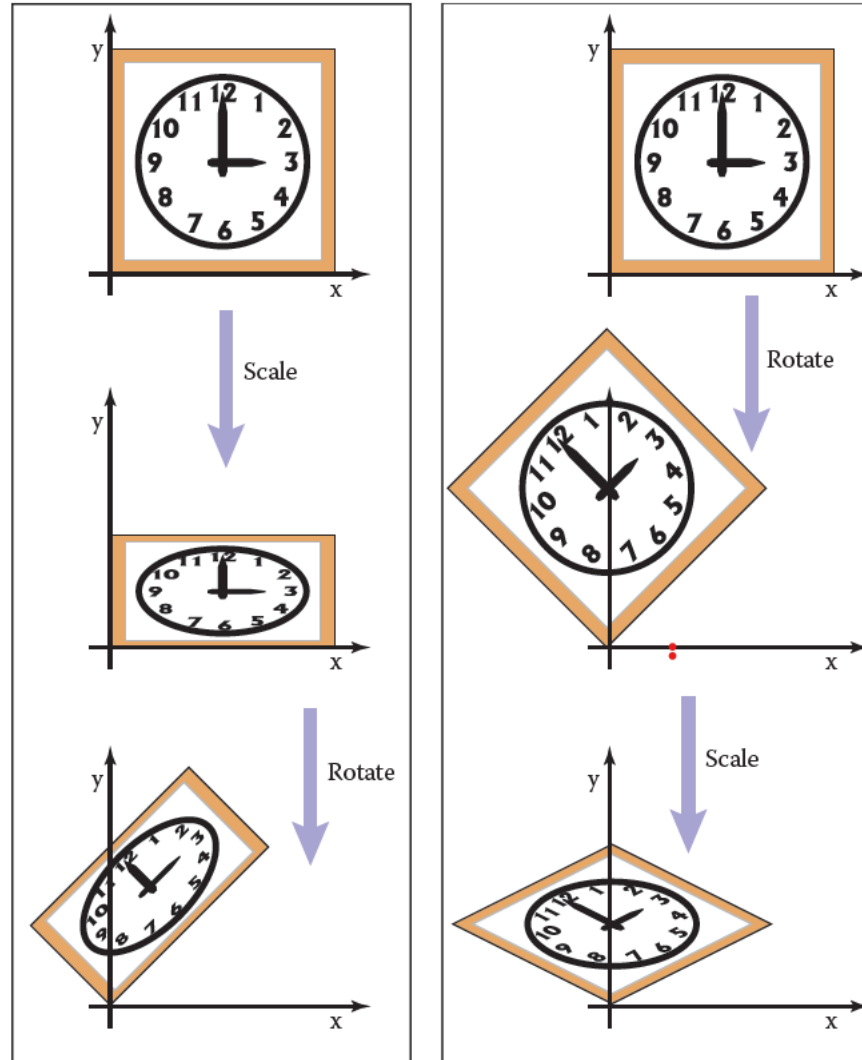


- We can write the sequence as **$q = CBAp$**
 - Note that matrix on the right is the first applied
 - Mathematically, the following are equivalent

$$q = CBAp = C(B(Ap))$$

- If we have many points to transform \Rightarrow
 $M = CBA$
 $q = Mp$

Order of Transformation matters



General Rotation About the Origin

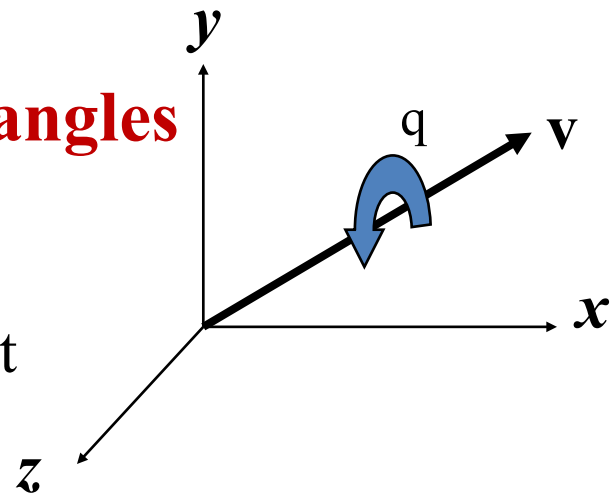
A rotation angle of θ about the origin can be decomposed into the concatenation of rotations about the x , y , and z axes

$$R(\theta) = R_z(\theta_z) R_y(\theta_y) R_x(\theta_x)$$

θ_x , θ_y , and θ_z are called the **Euler angles**

Note that rotations do not commute

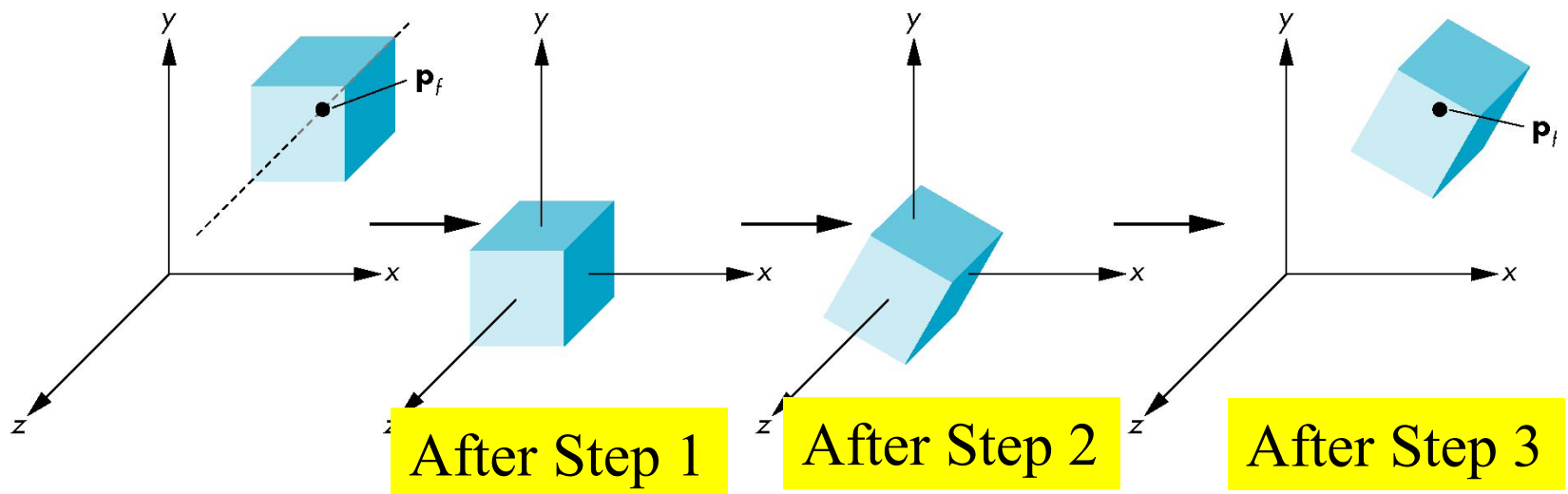
We can use rotations in another order but with different angles



Rotation About a Fixed Point other than the Origin

1. Move the object to the origin
2. Rotate
3. Move the object back to the fixed point

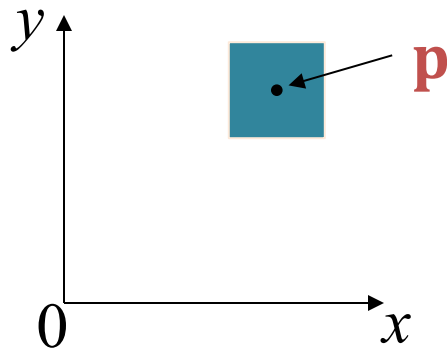
$$\Rightarrow \mathbf{M} = \mathbf{T}(\mathbf{p}_f) \mathbf{R}(\theta) \mathbf{T}(-\mathbf{p}_f)$$



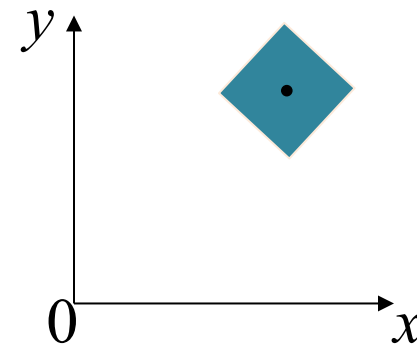
Rotation About a Fixed Point other than the Origin (cont.)

- A 2D example:

Objective: want to rotate a square 45 degrees about its own center, **p**.



Before rotation

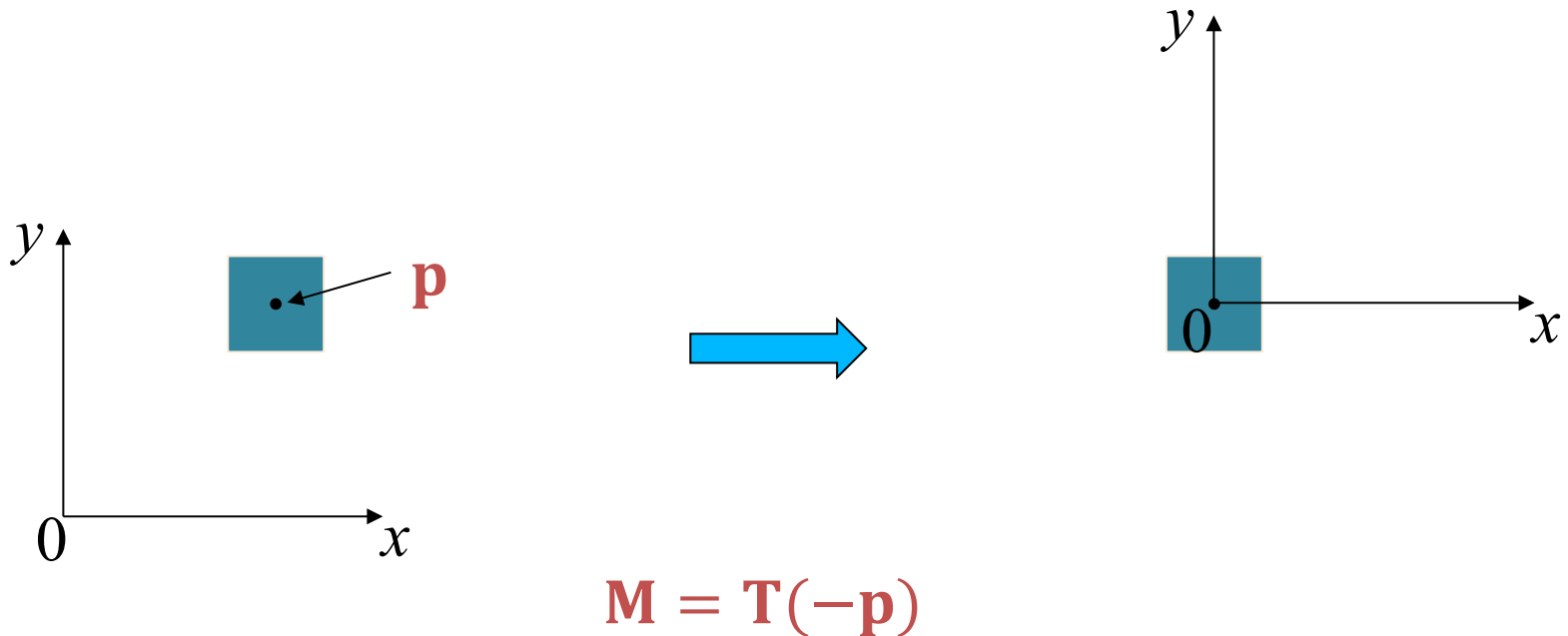


Output wanted after rotation

This is the same as rotating about the z-axis (pointing out of the page) in 3D.

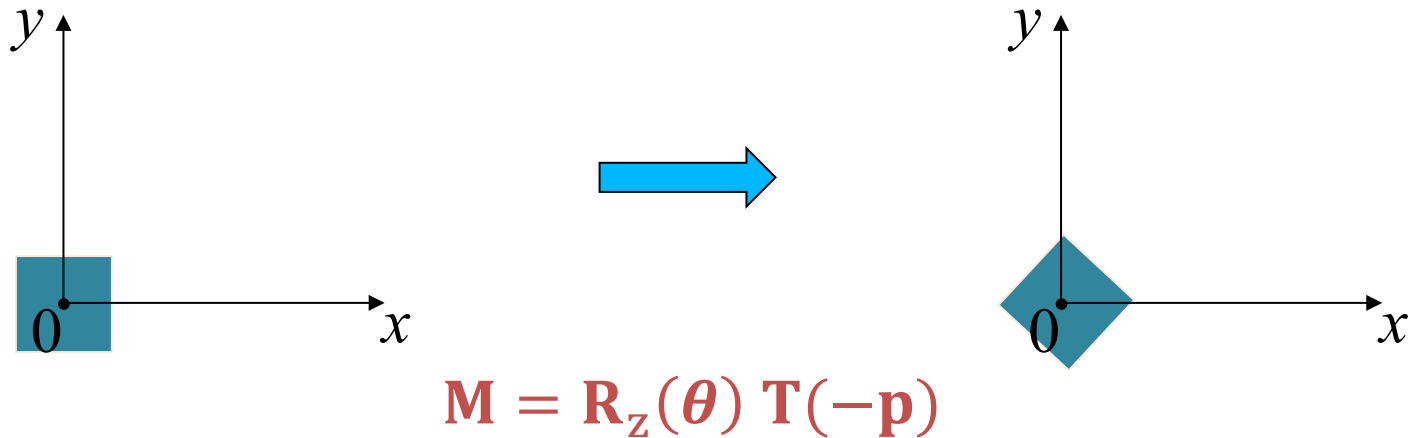
Rotation About a Fixed Point other than the Origin (cont.)

- Our aim is to construct a matrix \mathbf{M} so that when the four vertices of the square are multiplied by it, we get the desired output.
- **Step 1:** apply a translation so that the origin is at \mathbf{p} .



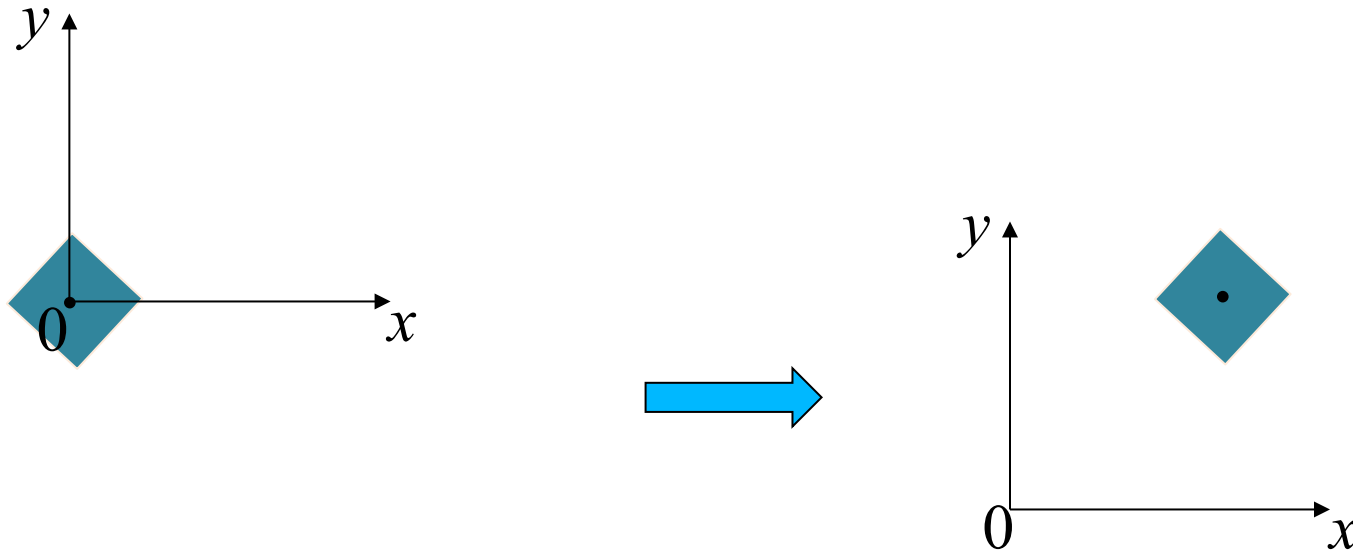
Rotation About a Fixed Point other than the Origin (cont.)

- **Step 2:** apply a 45-degree rotation about the z-axis at the origin.



Rotation About a Fixed Point other than the Origin (cont.)

- **Step 3:** move the origin back to where it was before.



$$\mathbf{M} = \mathbf{T}(\mathbf{p}) \mathbf{R}_z(\theta) \mathbf{T}(-\mathbf{p})$$

A Few Common Transformations

- **Rigid transformation:** The 4×4 matrix has the form:

$$\begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where R is a 3×3 rotation matrix and $\mathbf{t} \in \mathbb{R}^{3 \times 1}$ is a translation vector. Rigid transformation preserves everything (*angle* (this means the *shape*), *length*, *area*, etc.,)

- **Similarity transformation:** The matrix has the form:

Large (or small) s
values enlarge (or
diminish) the object

$$\begin{bmatrix} sR & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \text{ or } \begin{bmatrix} R & \mathbf{t} \\ \mathbf{0}^T & s' \end{bmatrix}$$

Small (or large) s'
values enlarge (or
diminish) the object

where $s, s' \neq 1$. Similarity transformation preserves *angle*, ratios of *lengths* and of *areas*.

A Few Common Transformations (cont.)

- **Affine transformation:** The 4×4 matrix has the form:

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

where A can be any 3×3 non-singular matrix and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector. Affine transformation preserves *parallelism*, ratios of lengths.

- **Perspective transformation:** The matrix can be any non-singular 4×4 matrix. Perspective transformation matrix preserves *cross ratios* (i.e., ratio of ratios of lengths).

A Few Common Transformations (cont.)

- Rigid transformation is equivalent to a change in coordinate frames. It has 6 **degrees of freedom (dof)** i.e., 3 rotations + 3 translations (along each of the three axes)
- Similarity transformation has 7 *dof* (an additional scaling)
- Affine transformation has 12 *dof*
 - 3 rotations + 3 translations + 3 scaling + 3 shear

slido



Choose the correct option

① Start presenting to display the poll results on this slide.

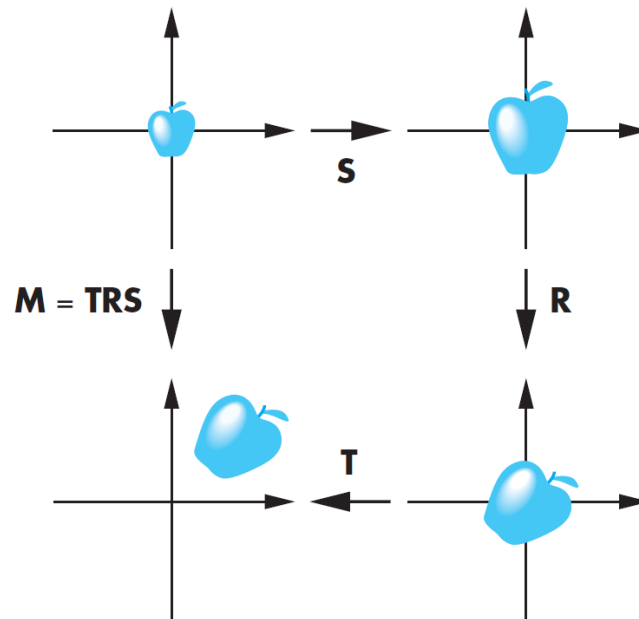


**What does the given
Transformation matrix do?**

① Start presenting to display the poll results on this slide.

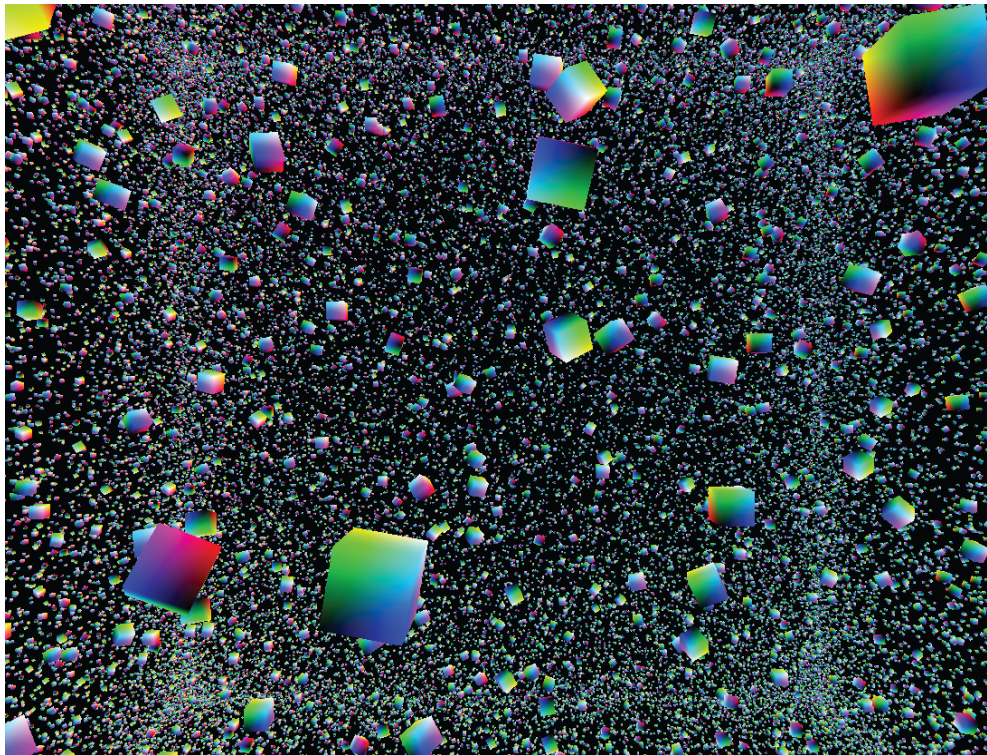
Instancing

- In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size
- We apply an *instance transformation* to its vertices to
 - Scale
 - Orient
 - Locate



Instancing

- *Instancing* provides a mechanism for telling the graphics card to render multiple copies of an object using only a single OpenGL call.



Instancing: 100,000 animated cubes

Using `glDrawArraysInstanced()` instead of `glDrawArrays()`

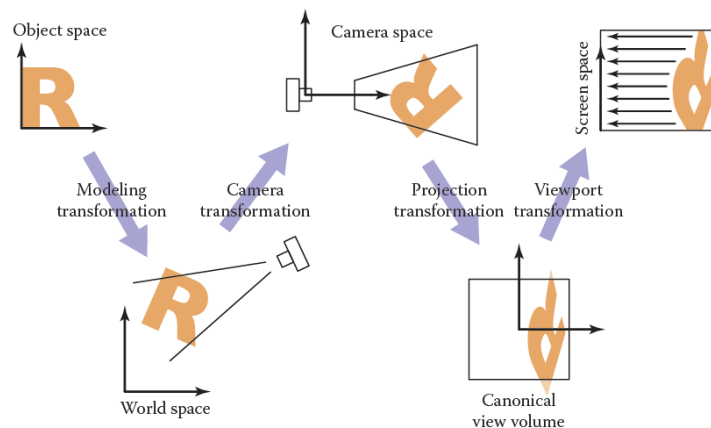
```
glDrawArraysInstanced(GL_TRIANGLES, 0, 36, 100000);
```

number of instances to be drawn

Frames in OpenGL

We had considered the following coordinate systems

1. Object (or model) coordinates
2. World coordinates
3. Eye (or camera) coordinates
4. Clip coordinates
5. Normalised device coordinates
6. Window (or screen) coordinates



Change of Coordinate Frame

In OpenGL, we specify our geometry using the coordinate system or frame that is natural for the model, which is known as the **object** or **model frame**.

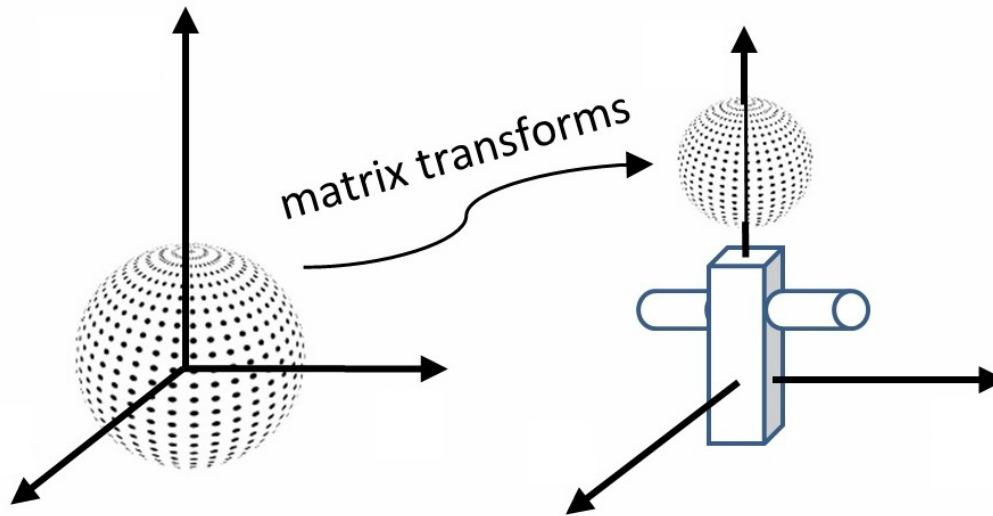
Models are then brought into the **world frame**

At some point, we want to know how these objects appear to the camera. It is natural at that point to convert from the world frame to the **camera** or **eye frame**

The conversion from the object frame to the eye frame is done by the model-view matrix.

Object (Local) and World Space

- Object (Local or Model) space:
 - *The space in which a model is defined, usually centered at the origin.*
- World space:
 - *The space in which a simulated scene is built*

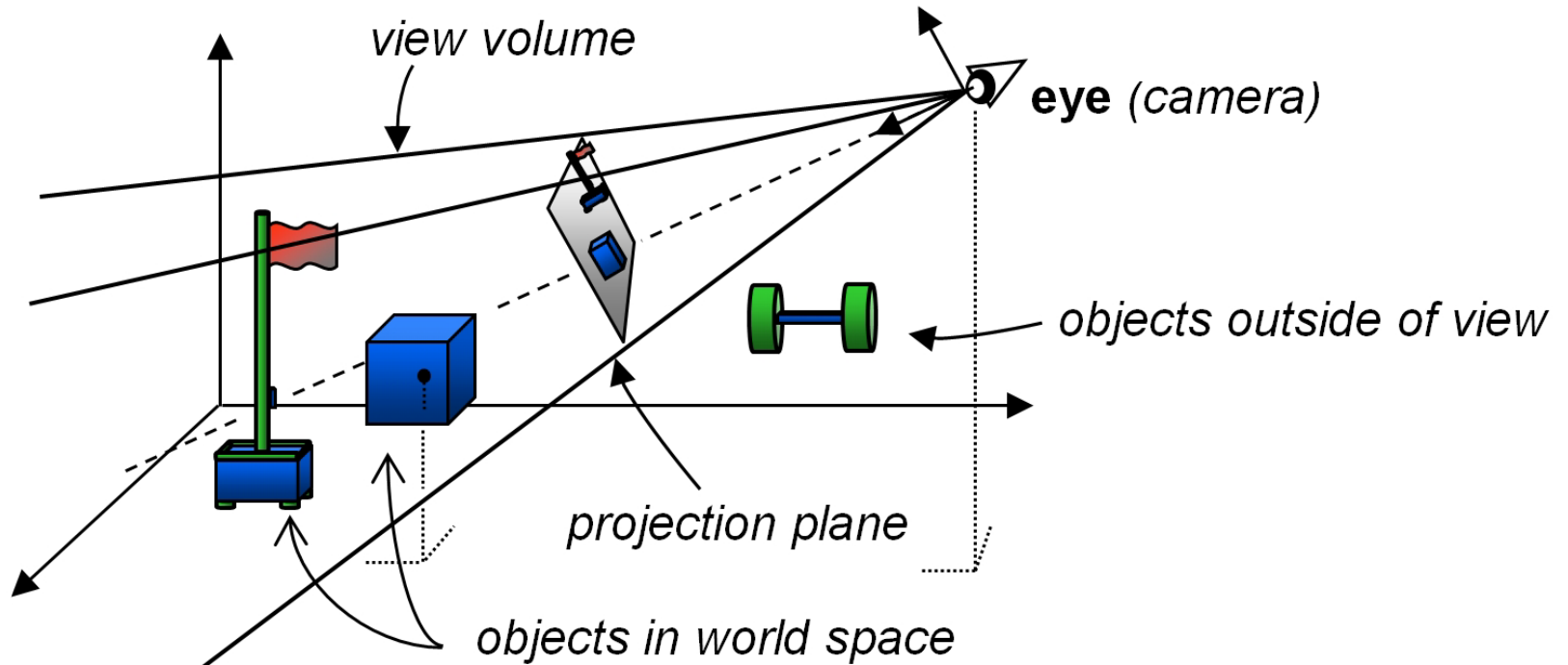


The *Model* matrix positions and orients the object in the world coordinate space. Each model has its own model matrix.

The matrix transforms (concatenated) that place an object in world space is called its Model matrix, or M

Camera (or View or Eye) Space

- World space as seen from a simulated camera or “eye”.*



Camera (or View or Eye) Space

- The View matrix moves and rotates the models in the world to simulate the effect of a camera at a desired location.
- OpenGL camera exists at location (0,0,0) and faces down the negative Z axis.
- To simulate the appearance of that camera being moved a certain way, we will need to move the objects themselves in the opposite direction.

Transformations in Graphics pipeline

We had considered the following coordinate systems

Can be
combined in
model-view
transform

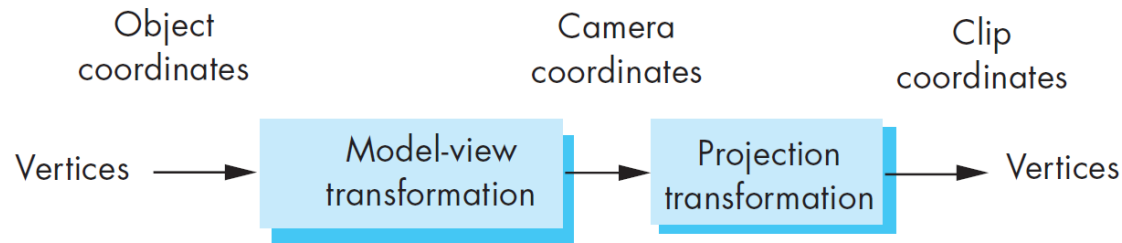
1. Object (or model) coordinates
2. World coordinates
3. Eye (or camera) coordinates

Affine
transform

4. Clip coordinates
5. Normalised device coordinates
6. Window (or screen) coordinates

The model-view transformation brings representations of geometric objects from application or model frame to the camera frame.

The projection matrix will both carry out the desired projection and also change the representation to clip coordinates.

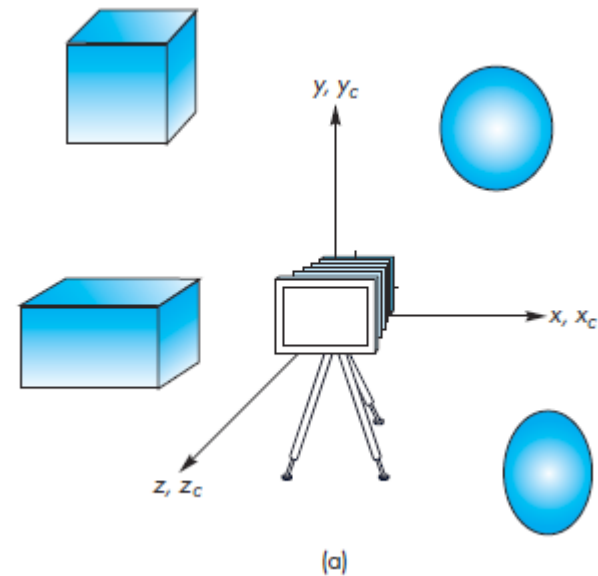


Camera and Object Frames

Camera and object frame in default positions

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

↑
model-view matrix



the camera is initially pointing in the negative z-direction

Note that if we do not model with predefined objects or apply any transformations before we specify our geometry, object and world coordinates are the same.

Camera and Object Frames

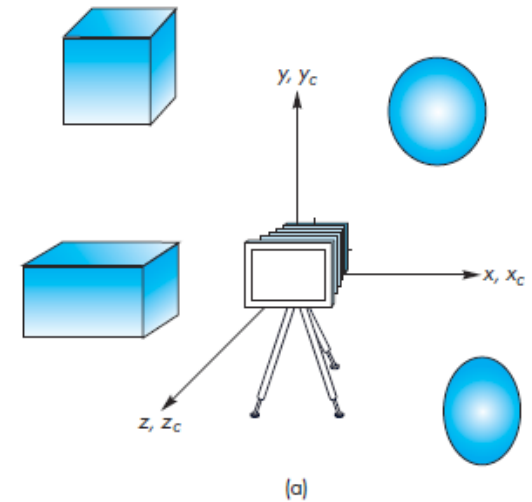
In most applications, we model our objects as being located around the origin, so a camera located at the default position with the default orientation does not see all the objects in the scene.

Thus, either:

- we must move the camera away from the objects that we wish to have in our image

Or

- the objects must be moved in front of the camera.



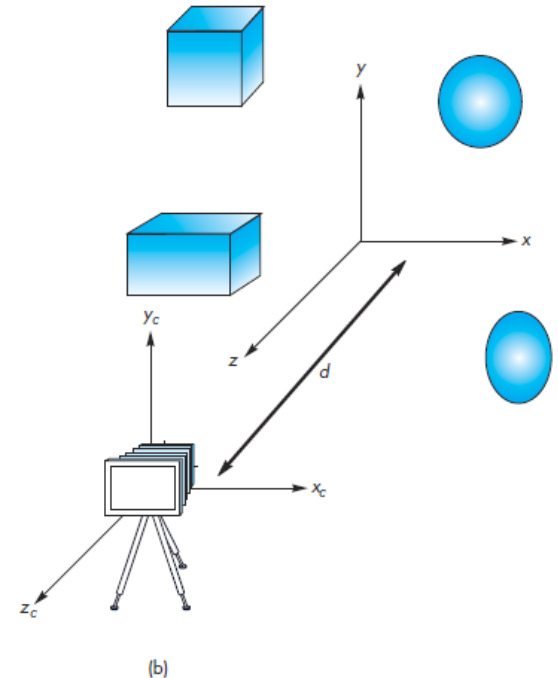
These are equivalent operations, as either can be looked at as positioning the frame of the camera with respect to the frame of the objects.

Positioning Object frame relative to the Camera frame

Camera frame is fixed, we are placing object frame relative to the camera frame.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation matrix ' \mathbf{A} ' moves a point (x, y, z) in the world frame to the point $(x, y, z - d)$ in the camera frame

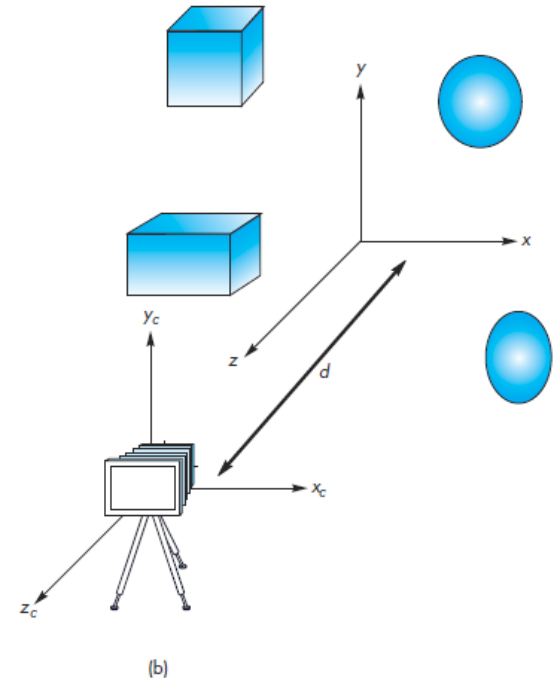


Moving the Camera Vs Moving the Objects

Camera frame is fixed, we are placing world frame relative to the camera frame.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

How did we derive \mathbf{A} (model-view matrix)?



Positioning World frame relative to the Camera frame

Camera frame is fixed, we are placing world frame relative to the camera frame.

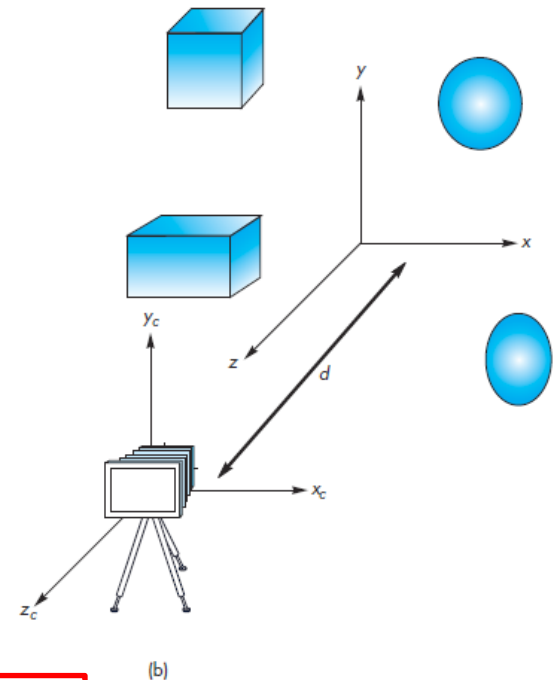
$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recall?

Translation matrix

- $x' = x$
- $y' = y$
- $z' = z + (-d)$

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



Moving the Camera

Moving the camera away from its initial position at the origin of the object frame.

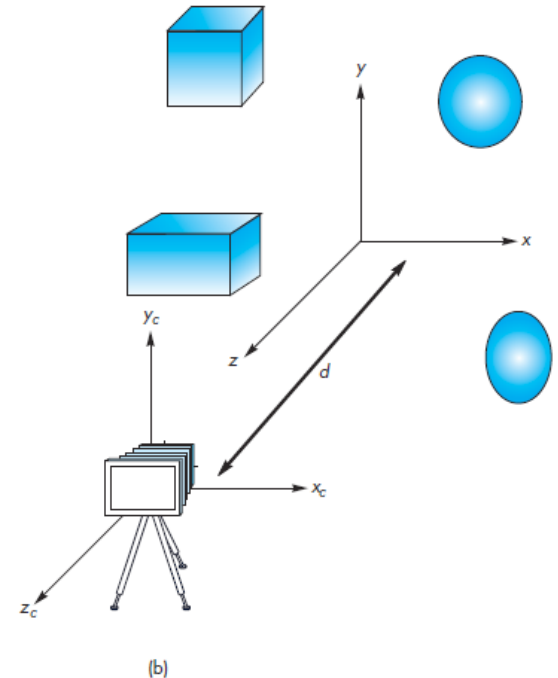
$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Recall? ... changing coordinate frames

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

where,

$$\mathbf{T} = (\mathbf{M}^T)^{-1}$$

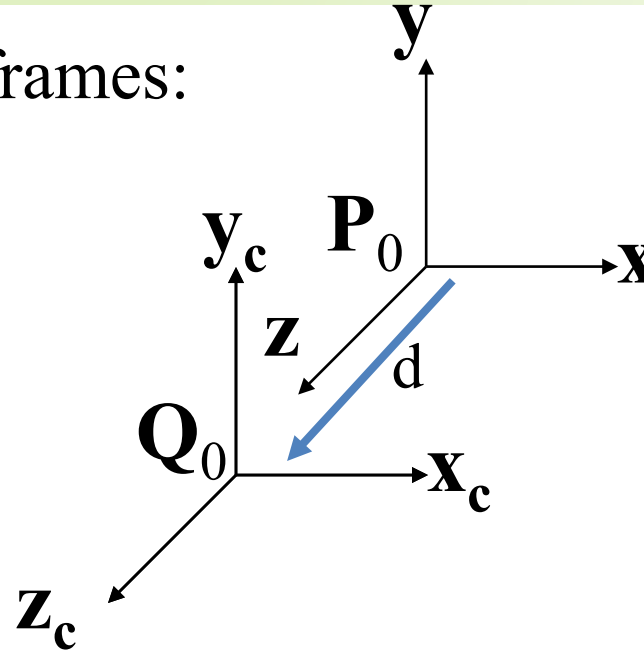


Recall: Change of Coordinate Frames

Consider two coordinate frames:

$(\mathbf{P}_0, \mathbf{x}, \mathbf{y}, \mathbf{z})$

$(\mathbf{Q}_0, \mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$



Initially both coordinate frames shared the same origin (0,0,0), then we moved the camera frame to (0,0,d)

- We can represent $(\mathbf{Q}_0, \mathbf{x}_c, \mathbf{y}_c, \mathbf{z}_c)$ in terms of $(\mathbf{P}_0, \mathbf{x}, \mathbf{y}, \mathbf{z})$

$$\mathbf{x}_c = \gamma_{11}\mathbf{x} + \gamma_{12}\mathbf{y} + \gamma_{13}\mathbf{z}$$

$$\mathbf{y}_c = \gamma_{21}\mathbf{x} + \gamma_{22}\mathbf{y} + \gamma_{23}\mathbf{z}$$

$$\mathbf{z}_c = \gamma_{31}\mathbf{x} + \gamma_{32}\mathbf{y} + \gamma_{33}\mathbf{z}$$

$$\mathbf{Q}_0 = \gamma_{41}\mathbf{x} + \gamma_{42}\mathbf{y} + \gamma_{43}\mathbf{z} + \mathbf{P}_0$$

$$\mathbf{x}_c = \mathbf{z}$$

$$\mathbf{y}_c = \mathbf{y}$$

$$\mathbf{z}_c = \mathbf{z}$$

$$\mathbf{Q}_0 = d\mathbf{z} + \mathbf{P}_0$$

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

Moving the Camera

Moving the camera away from its initial position at the origin of the object/world frame.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

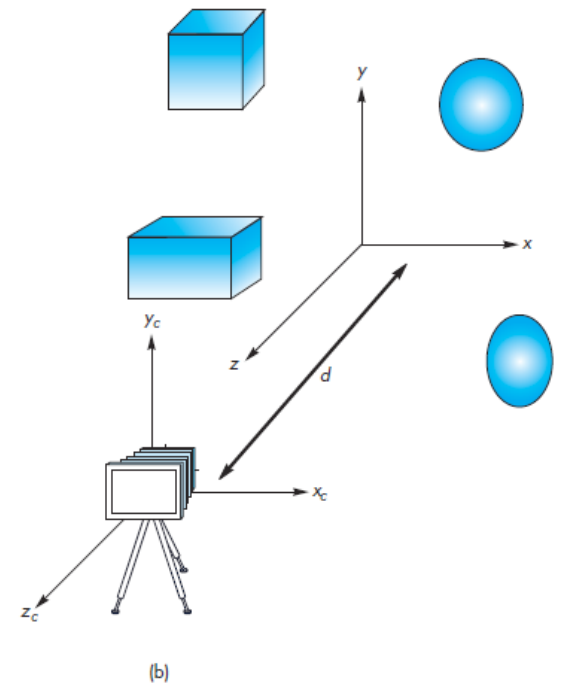
Recall? ... changing coordinate frames

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

where,

$$\mathbf{T} = (\mathbf{M}^T)^{-1}$$

$$\text{where, } \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & d & 1 \end{bmatrix}$$



‘**b**’ is the representation of a point/vector w.r.t the camera coordinates

‘**a**’ is the representation of a point/vector w.r.t the world coordinates

Positioning Camera frame relative to the World frame

World frame is fixed, we are placing camera frame relative to the world frame.

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

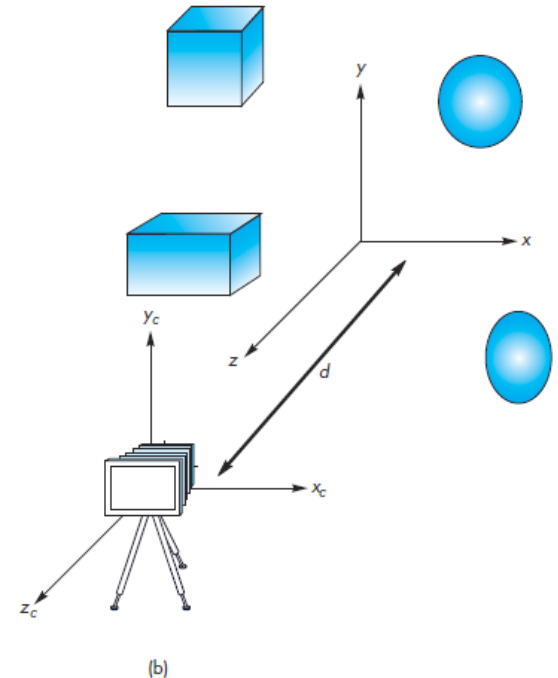
This matrix \mathbf{T} takes a point $(0, 0, d)$ in the object/world frame, whose representation is:

$$\mathbf{p} = [0 \ 0 \ d \ 1]^T$$

to

$$\mathbf{p}' = [0 \ 0 \ 0 \ 1]^T$$

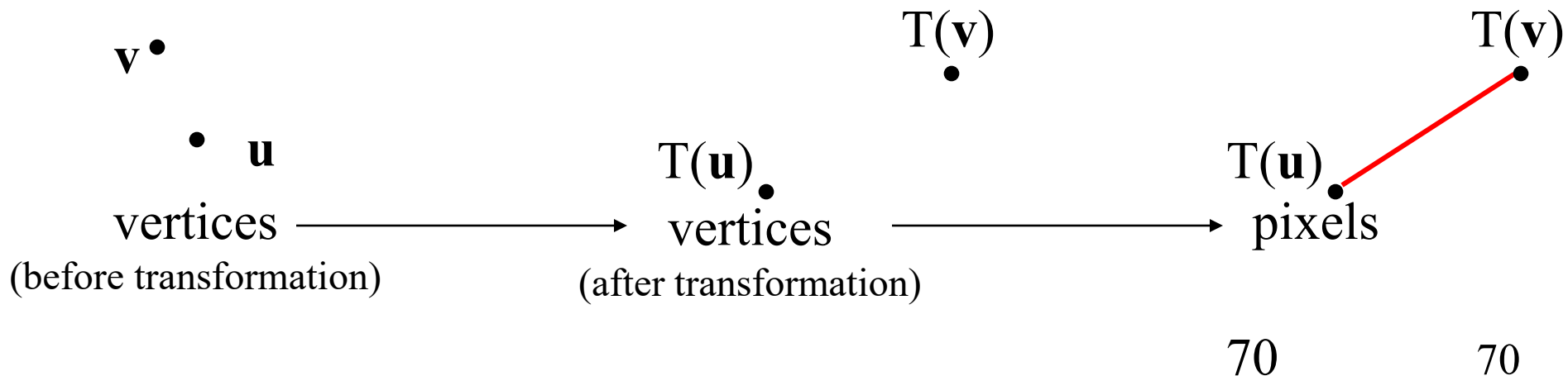
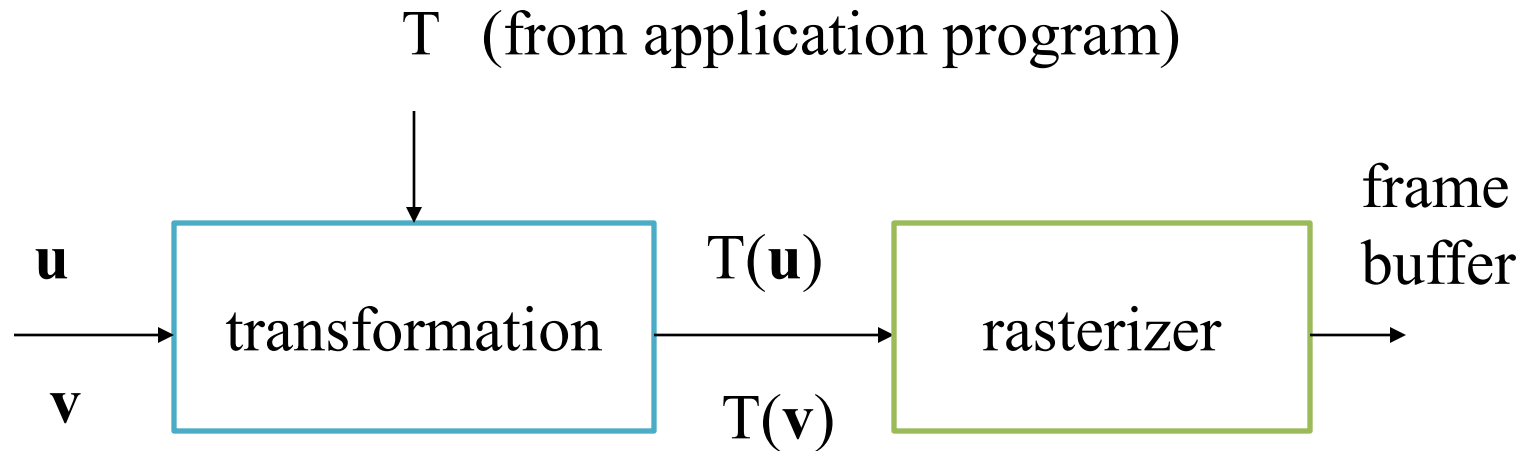
i.e., the origin in the camera frame



The World and Camera Coordinate Frames

- Changes in coordinate frame are then defined by 4×4 matrices
- In OpenGL, the base frame that we start with is the **world frame**
- Eventually we represent entities in the **camera frame** by changing the world representation using the **model-view matrix**
- Initially these frames are the same (i.e., **$\mathbf{M}=\mathbf{I}$**)

Pipeline Implementation



Further Reading

“Interactive Computer Graphics – A Top-Down Approach with Shader-Based OpenGL” by Edward Angel and Dave Shreiner, 6th Ed, 2012

- Sec 3.7 *Affine Transformations* (all subsections)
- Sec 3.8 *Translation, Rotation, and Scaling*
- Sec 3.9 *Transformations in Homogeneous Coordinates*
- Sec 3.10 *Concatenation of Transformations*

“Fundamentals of Computer Graphics , Steve Marschner and Peter Shirley, 4th Edition”

- Chapter#06, Transformation Matrices

“Computer Graphics Programming in OpenGL with C++, by V. Scott Gordon and John Clevenger”, 2nd Edition

- Sec 4.6.1 Instancing