Transformations

Computer Graphics
Instructor: Sungkil Lee

Today

- **Coordinate systems and frames for affine spaces**
  - Homogeneous coordinates
- **Affine transformations**
  - Rotation, translation, scaling, shear
- **How to build arbitrary transformation matrices**
  - from simple standard transformation matrices

Prerequisites

Linear Independence, Dimension

- **Linear independence**
  - A set of vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n$ is ***linearly independent*** if

$$\alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_n \boldsymbol{v}_n = 0, \qquad \text{iff } \alpha_1 = \alpha_2 = \cdots = 0$$

  - If a set of vectors are ***linearly independent***, we cannot represent one in terms of the others . Otherwise, 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

Basis and Representation

- **Basis**
  - In an n-dimensional space, any set of n linearly independent vectors form a ***basis*** for the space; such sets are not unique.
  - Given a basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n\}$, any vector $\boldsymbol{v}$ can be written as

$$\boldsymbol{v} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \cdots + \alpha_n \boldsymbol{v}_n \,, \quad \text{where the } \{\alpha_i\} \text{ are unique.}$$

- **Representation**
  - The list of scalars $\{\alpha_i\}$ is the representation of v with respect to $\{\boldsymbol{v}_i\}$.
  - We can write the representation as a column vector of scalars.

$$\boldsymbol{a} = [\alpha_1 \alpha_2 \ \dots \alpha_n]^{\mathrm{T}} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \dots \\ \alpha_n \end{bmatrix}$$

Coordinate-Free Geometry

Coordinate-Free Geometry

- **Coordinate-free geometry**
  - Points exist in space regardless of any reference or coordinate system.
  - Thus, we do not need a coordinate system to specify a point or a vector.
  - Most of geometric results are independent of the coordinate system

- **Example: Euclidean geometry**
  - Two triangles are identical if two corresponding sides and the angle between them are identical

Coordinate-Free Geometry

- **This fact may seem counter to your experience:**
  - When we learned simple geometry in high school, most of us started with a Cartesian approach; e.g., point p is at locations in space (x, y, z)
  - This approach is nonphysical, because points exist regardless of a coordinate system; as covered in middle-school math.
  - See Figures 3.6 and 3.7 to understand the difference.

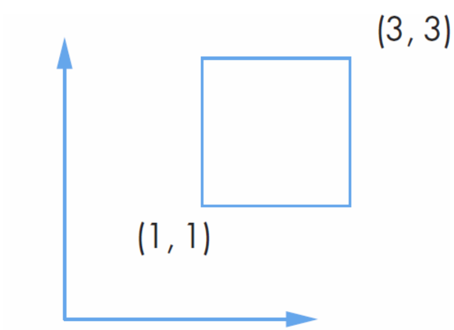


FIGURE 3.6 Object and coordinate system.

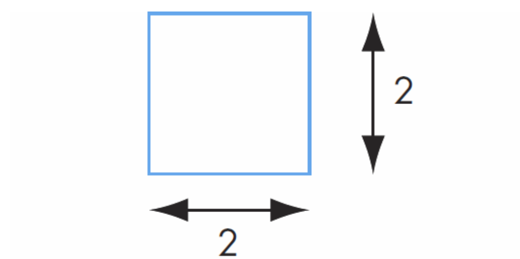


FIGURE 3.7 Object without coordinate system.

Coordinate-Free Geometry

- **We may find it inconvenient to use coordinate-free geometry.**
  - We may need to refer a specific point as "**that blue point over there**."
  - But, it is important to understand that the fundamental geometric relationships are preserved without coordinate systems.
    - The square is still there, or orthogonal lines are still orthogonal, and distances between points remain the same.

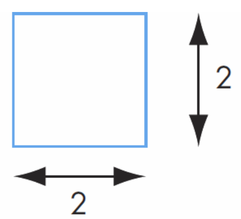


FIGURE 3.7 Object without coordinate system.

  - This reference problem is later solved by coordinate systems and frames.

Coordinate Systems and Frames

Frame of Reference

- **So far, we have been able to work with geometric entities without using any *frame of reference*.**

- **We need a frame of reference to relate points and objects to our physical world.**
  - For example, where is a point?
    - Hard to answer without a reference system
  - Coordinate system examples
    - World coordinates, camera coordinates, screen-space coordinates, …

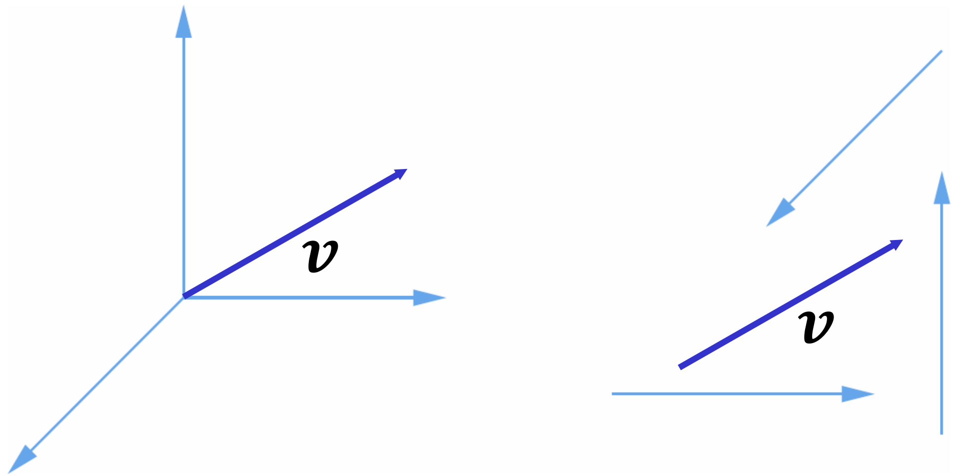
Coordinate Systems

- **An example:**
  - $\boldsymbol{a}$ is a particular representation of a vector $\boldsymbol{v}$, with respect to the basis vectors $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$ that define a *coordinate system*.

$$\boldsymbol{v} = 2\boldsymbol{v}_1 + 3\boldsymbol{v}_2 - 4\boldsymbol{v}_3$$
$$\boldsymbol{a} = [\,2 \;\; 3 \; -4\,]^{\mathrm{T}}$$

Coordinate Systems

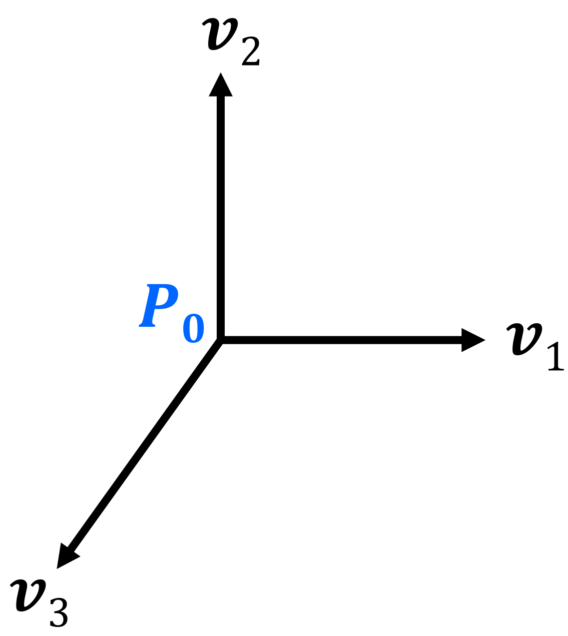
- **Problem**
  - Which is correct?



  - Both are correct, because vectors have no fixed locations.

Frames

- **A coordinate system is insufficient to represent points,**
  - because points are not defined in a coordinate system.
- **If we work in an *affine space*, we can add a single point, the *origin*, to the basis vectors to form a *frame.***



Frames: Representation

- **Suppose a frame determined by** $(P_0, \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3)$.
  - Within this frame, every vector and point can be written as

$$\boldsymbol{v} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \alpha_3 \boldsymbol{v}_3,$$
$$P = P_0 + \beta_1 \boldsymbol{v}_1 + \beta_2 \boldsymbol{v}_2 + \beta_3 \boldsymbol{v}_3$$

  - They appear to have the similar representations, yet a vector has no position.

Frames: Representation

- **If we define** $0 \bullet P$ **= 0 and** $1 \bullet P = P$ **then we can write them,**

$$\boldsymbol{v} = \alpha_1 \boldsymbol{v}_1 + \alpha_2 \boldsymbol{v}_2 + \alpha_3 \boldsymbol{v}_3 + \mathbf{0} = [\alpha_1\ \alpha_2\ \alpha_3\ 0][\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ P_0]^{\mathrm{T}}$$
$$P = \beta_1 \boldsymbol{v}_1 + \beta_2 \boldsymbol{v}_2 + \beta_3 \boldsymbol{v}_3 + \boldsymbol{P}_{\mathbf{0}} = [\beta_1\ \beta_2\ \beta_3\ 1][\boldsymbol{v}_1\ \boldsymbol{v}_2\ \boldsymbol{v}_3\ P_0]^{\mathrm{T}}$$

  - By this way, we can represent the vectors and points in a single representation.

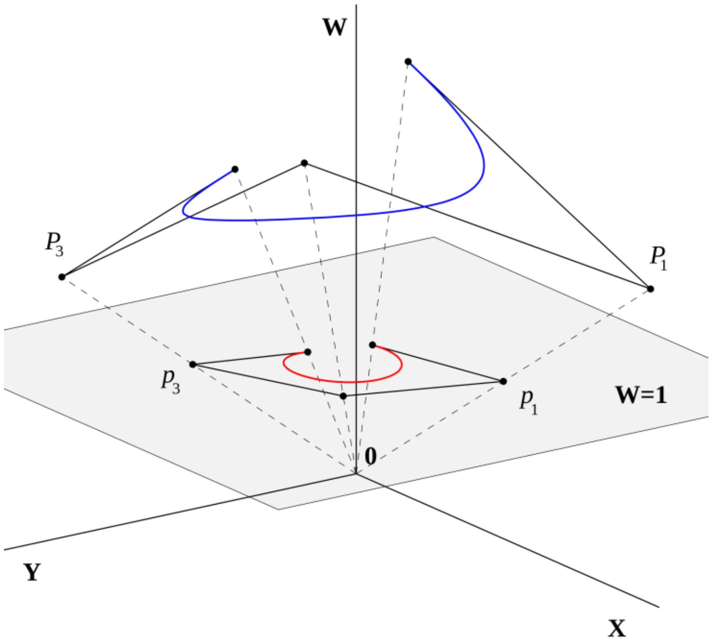
Homogeneous Coordinates Representations

- **4-D *homogeneous coordinate (HC) representation* represents vectors and points in 3D Cartesian coordinates (CC).**

$$p = [x' \; y' \; z' \; w]^{\mathrm{T}}$$

  - If $w = 0$, the representation refers to a vector.
  - If $w \neq 0$, the representation refers to a point in 3D.
  - If $w \neq 0$ and $w \neq 1$, we can return to a 3D point by dividing (x', y', z') by w.
    - Perspective division

- **More intuitive example**
  - 2D line in HC = 1D point in CC
  - 1D point in CC is found at w=1 in HC



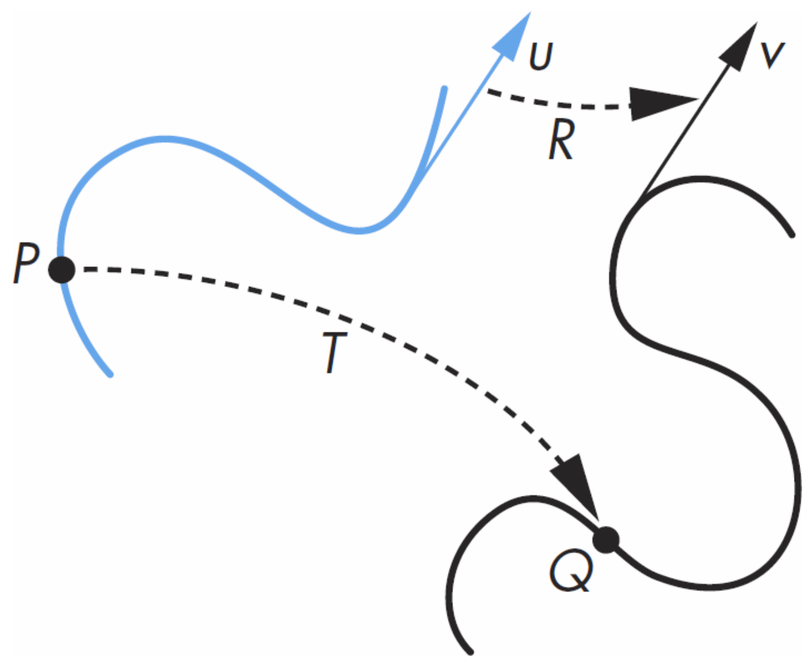
HCs in Computer Graphics

- **Homogeneous coordinates are keys to all computer graphics systems.**
  - All standard transformations (rotation, translation, and scaling) can be implemented with ***matrix multiplications using 4x4 matrices***.
  - Hence, graphics hardware pipeline is designed to work with ***4-D representations***.

General vs. Affine Transformations

General Transformations

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



$$v = R(u)$$

$$Q = T(P)$$

Affine Transformations

- **A matrix for the change of frames is 4×4 and specifies an affine transformation in homogeneous coordinates.**
  - Affine transformation is a function between affine spaces which preserves points, lines, and planes.
  - Every linear transformation is equivalent to a change of frames.

- **Affine transformations have 12 degrees of freedom (dof).**
  - Upper 4×3 elements of the matrix is defined by 3 dofs from translation, 3 dofs from rotation, 3 dofs from scaling, and 3 dofs from shear transformations.
  - 4 of the elements (the bottom row) in the matrix are fixed.

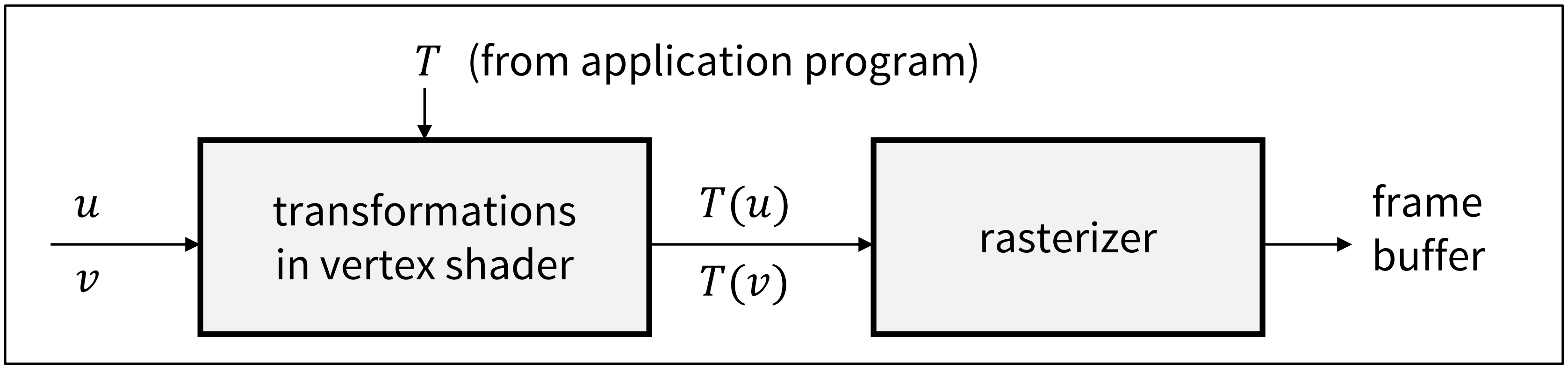
Affine Transformations

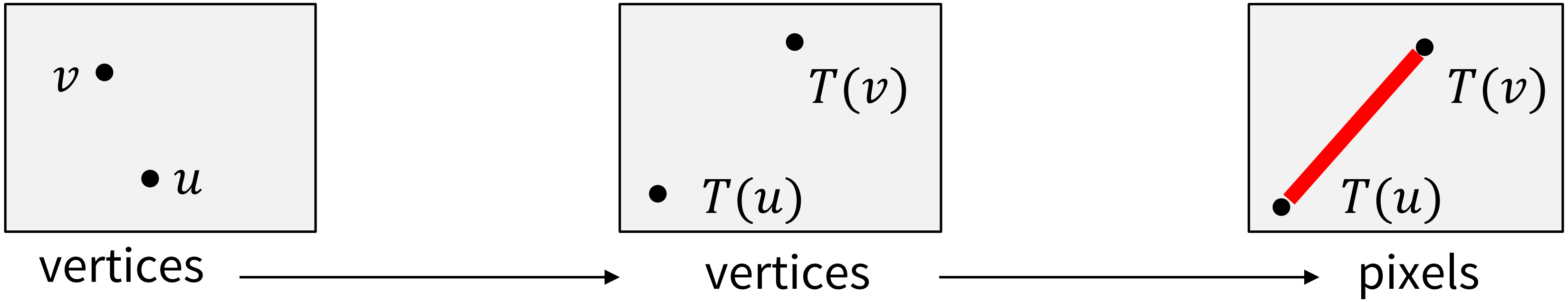
- **Affine transformation *preserves lines*.**
  - Lines will remain lines (not become curves or be broken into segments).
  - Characteristic of many physically important transformations
    - Rigid body transformations: translation, rotation
    - Scaling, shear

- **Importance in graphics:**
  - **We can only transform endpoints of line segments** and let implementation draw the line segment by interpolating transformed endpoints.
  - This allows us to realize pipeline approach.
    - Recall that we only transform vertices instead of lines.

Affine Transformations

- **Affine transformation *preserves lines*.**
  - This allows us to realize pipeline approach.





Standard (Affine) Transformations

Notation

- **We will work with both coordinate-free representations and representations within a HC frame.**

P,Q,R: **points in an affine space**

u,v,w: **vectors in an affine space**

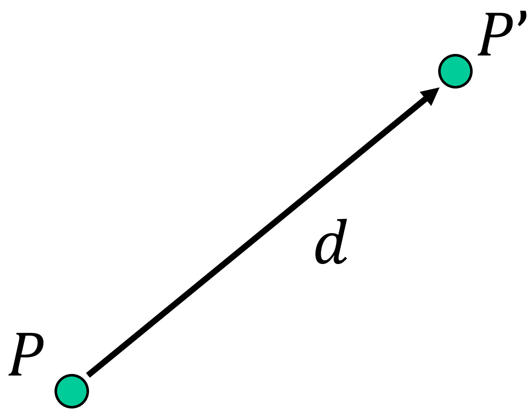
α,β,γ: **scalars**

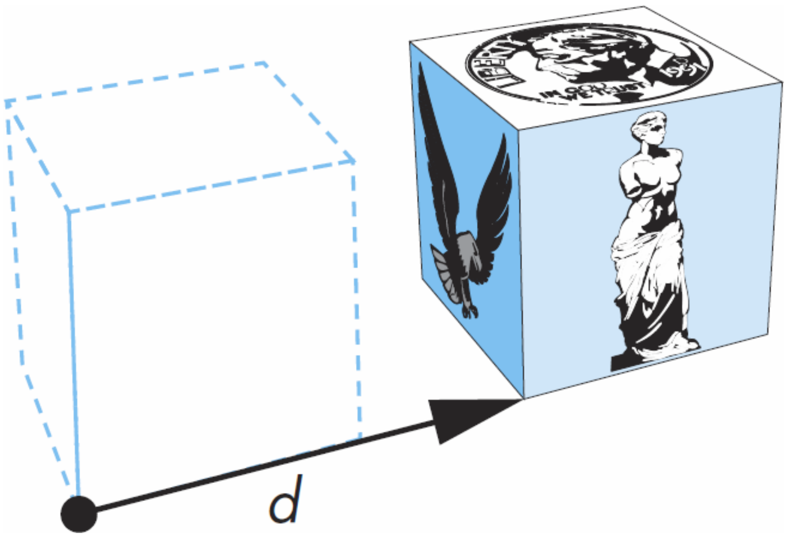
$\boldsymbol{p},\boldsymbol{q},\boldsymbol{r}$: **4-D representations of points in HC**

$\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}$: **4-D representations of vectors in HC**

Translation

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





- **Displacement determined by a vector** d
  - 3 degrees of freedom
  - $P' = P + d$

Translation Using Representations

- **Using the homogeneous coordinate representation in some frame**

$$\boldsymbol{p} = [\,x \;\; y \;\; z \;\; 1]^{\mathrm{T}}$$
$$\boldsymbol{p}' = [x' \;\; y' \; z' \;\; 1]^{\mathrm{T}}$$
$$\boldsymbol{d} = [d_x \; d_y \; d_z 0]^{\mathrm{T}}$$

- **Hence $p' = \boldsymbol{p} + \boldsymbol{d}$ or**

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

- **Note that this expression is in 4-Ds and expresses point-vector addition.**

Translation Matrix

- **We can also express translation using a 4×4 matrix T in homogeneous coordinates $p' = \mathrm{T}p$ where**

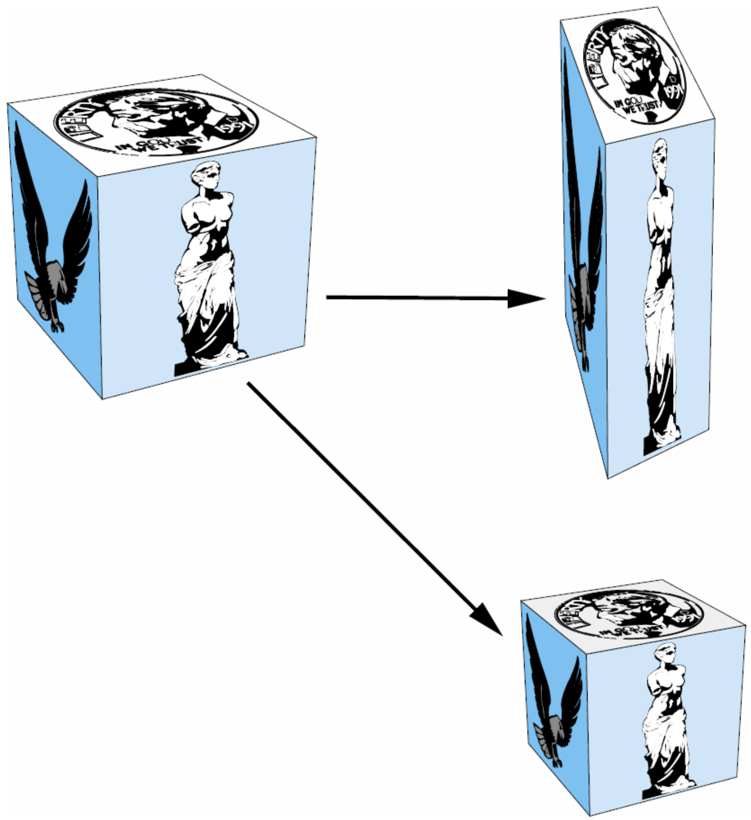
$$\boldsymbol{T} = \boldsymbol{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- **This form is better for implementation because**
  - all affine transformations can be expressed in this way (4×4 matrix)
  - and multiple transformations can be concatenated together by multiplying them together.

Scaling

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

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



Scaling Matrix

- **In homogeneous coordinates,**

$$\boldsymbol{p}' = \mathbf{S}\boldsymbol{p}$$

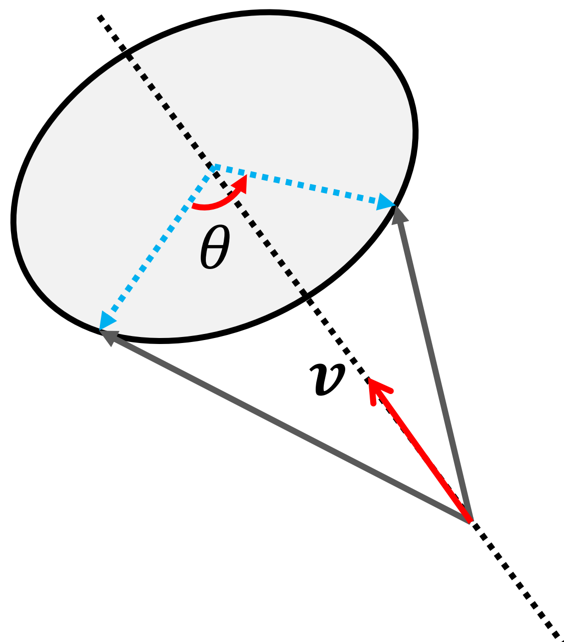
where

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

Rotation

- **Generally, rotation transformation can be described by the rotation angle θ and its revolution axis v.**

$$\boldsymbol{p}' = \boldsymbol{R}(\theta)\boldsymbol{p}$$



Rotation Matrix

- **Rotation matrix $R(\theta)$ revolving axis v is given as:**

$$\begin{bmatrix} v_x v_x(1-c)+c & v_x v_y(1-c)-v_z s & v_x v_z(1-c)+v_y s & 0 \\ v_x v_y(1-c)+v_z s & v_y v_y(1-c)+c & v_y v_z(1-c)-v_x s & 0 \\ v_x v_z(1-c)-v_y s & v_y v_z(1-c)+v_x s & v_z v_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where $\mathrm{c} = \cos\theta,\ s = \sin\theta$

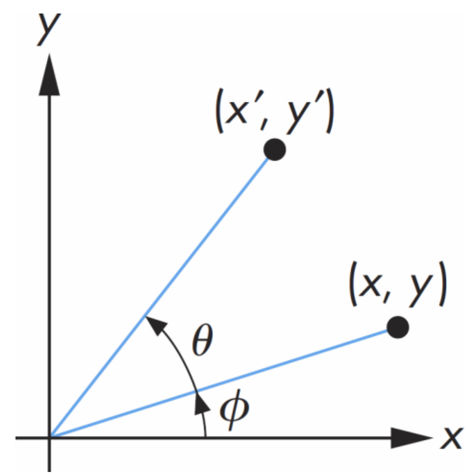
- This formulation is dervied using Quaternion (an extension of complex numbers with three imaginary numbers).
- Though, we do not prove this, because a rigorous proof for this goes far beyond the undergraduate level.

Rotation Matrix

- **Rotation about z axis in 3D**
  - With z-axis ($v_x = 0, v_y = 0, v_z = 1$), the formulation is reduced to the well-known form.
  - equivalent to 2-D rotation in planes of constant z, like slicing 3D into multiple plane slices at height z and rotating in each such plane.

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

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



Rotation Matrix

- **Similarly, rotation matrix along x- and y-axes are:**

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

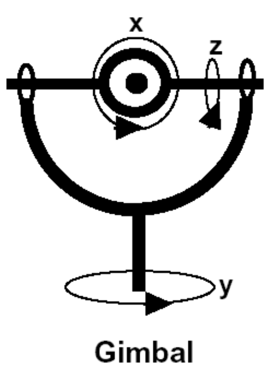
$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

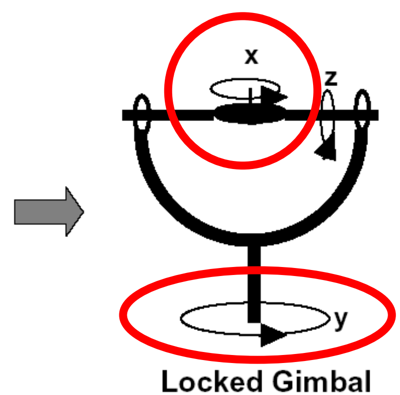
Rotation Matrix by Euler Angles

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

  - $\mathbf{R}(\theta) = \mathbf{R}_x(\theta_x)\mathbf{R}_y(\theta_y)\mathbf{R}_z(\theta_z)$, where $\theta_x, \theta_y, \theta_z$ are called the Euler angles.

  - However, do not apply the rotation matrices with the concatenated form of rotation matrices of Euler angles, because such a rotation may cause a gimbal lock problem in some cases.
  - **Gimbal lock**: when two axes effectively line up, a degree of freedom is lost.





Standard 2D Transformation Matrices

2D Transformation in 4x4 Matrix

- **Use 4x4 matrices instead of 2x2 or 3x3 matrices**
  - Graphics pipeline is optimized for 4x4 matrix, and thus, it is better to use 4x4 matrices even for 2D transformation.
  - It is consistent, when mixing with 3D transformations.

- **It is trivial to derive 2D transformations from 3D transformations.**
  - 2D translation: $\boldsymbol{T} = \boldsymbol{T}\left(d_x, d_y, 0\right)$
  - 2D Scaling: $\mathbf{S} = \mathbf{S}\left(s_x, s_y, 1\right)$
  - 2D rotation (with z-axis): $\mathbf{R} = \mathbf{R}_z(\theta)$

General Affine Transformations

Inverse Transformations

- **Basically, we can compute inverse matrices by general formulas.**
  - Though, the inverse operation is very costly, and also, can degrade precision in the computer arithmetic.
- **Alternatively, we can use simple geometric observations:**

$$\mathbf{T}^{-1}\left(d_x, d_y, d_z\right) = \mathbf{T}(-d_x, -d_y, -d_z)$$

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta) = \mathbf{R}^{\mathrm{T}}(\theta)$$

$$\boldsymbol{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$$

General Affine Transformations

- **Multiple transformations can be represented as *concatenation of transformation matrices.***

- **We can form arbitrary affine transformation matrices by multiplying rotation, translation, and scaling matrices together.**
  - The cost of forming a matrix $\mathbf{M}=\mathbf{ABCD}$ is insignificant, because the same transformation is applied to many vertices,

- **Note that matrix on the right is the first applied:**

$$p' = \mathrm{A}\big(\mathrm{B}(\mathrm{C}p)\big) = \mathbf{ABC}p$$

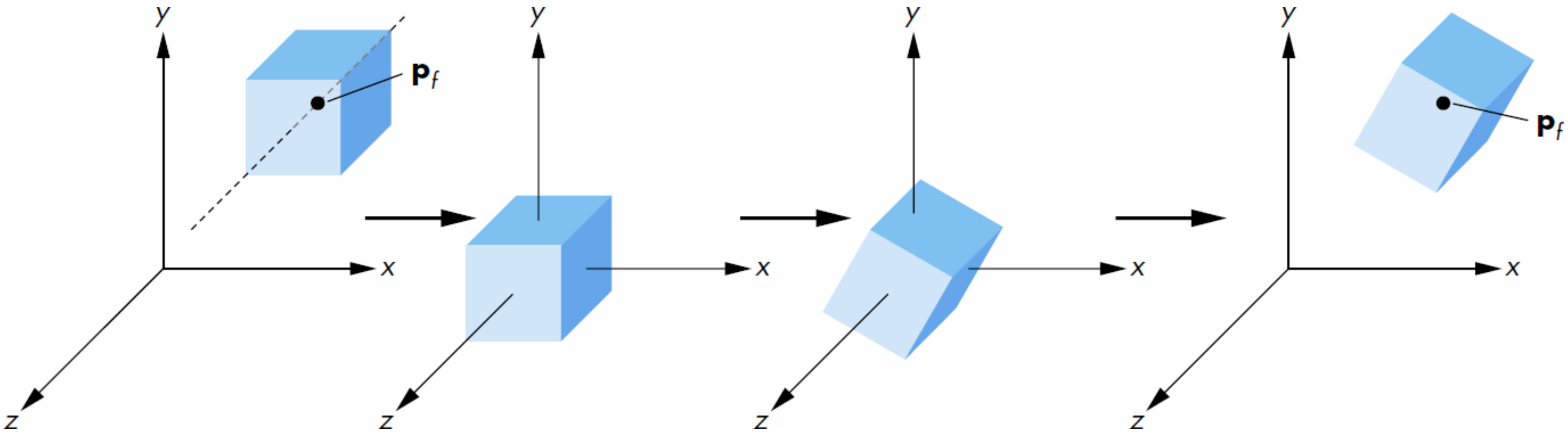
Rotation about Non-Origin Point

1) Move the fixed point to the origin

2) Rotate

3) Move the fixed point back

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



Instancing

- **In modeling, we often start with an 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, and locate somewhere.
  - This allows us to work with minimal geometric objects, while rendering many different objects.