CS482: Interactive Computer Graphics

Min H. Kim
KAIST School of Computing

AFFINE TRANSFORMATION

Points vs. vectors

- Vector $\vec{v}$:= motion between points in space
  - lives in a space we call R^3
  - has the structure of a linear/vector space.
  - addition and scalar multiplication have meaning
  - zero vector is no motion
  - cannot really translate motion

Points vs. vectors

- Point $\tilde{p}$:= a position in space
  - lives in a space we might call A^3
  - has the structure of a so-called affine space.
  - addition and scalar multiplication don't make sense
  - zero doesn't make sense
  - subtraction does make sense, gives us a vector

Points and frames

- Subtraction of points:

$$\tilde{p} - \tilde{q} = \vec{v}$$

- Moving a point with a vector:

$$\tilde{q} + \vec{v} = \tilde{p}$$

- Basis is three vectors

$$\vec{v} = \sum_i c_i \vec{b}_i$$

- How could we present <u>translations</u>?
  - Affine transform (4-by-4 matrix)
  - usage: transformation of objects and camera projection (3D → 2D)

Points and frames

- for affine space we will use a frame
  - start with a chosen origin point $\tilde{o}$
  - add to it a linear combination combination of vectors using coordinates c_i to get to any desired point $\tilde{p}$

Affine frame

- Movement of a point (original $\tilde{o}$ → a point $\tilde{p}$)

$$\tilde{p} = \tilde{o} + \vec{v}.$$

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}.$$

- Affine frame (made of three vectors and a point): $\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c}$.

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} = \vec{\mathbf{f}}^t$$

Defining an affine matrix

- point is specified with a 4-coordinates vector
  - four numbers
  - last one is always 1 $\begin{bmatrix} c_1 & c_2 & c_3 & 1 \end{bmatrix}^t$
  - … or 0. (and we get a vector)
- let's define an affine matrix as 4-by-4 matrix

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- we are transforming a point to another with an affine frame:

$$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c}.$$

Transforming a point

- affine transform

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}.$$

Min H. Kim (KAIST) CS482: Interactive Computer Graphics 9

Transforming a point

- for short

$\tilde{p} = \vec{\mathbf{f}}^t \mathbf{c} \Rightarrow \vec{\mathbf{f}}^t A \mathbf{c}$. where $\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} = \vec{\mathbf{f}}^t$

- transforming coordinate vectors (4 with a one as the fourth entry)

$$\begin{bmatrix} c'_1 \\ c'_2 \\ c'_3 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}.$$

Min H. Kim (KAIST) CS482: Interactive Computer Graphics 10

transforming a point

- Alternatively, transforming the basis vectors

$$\begin{bmatrix} \vec{b'}_1 & \vec{b'}_2 & \vec{b'}_3 & \tilde{o}' \end{bmatrix} = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

- This transformation is to apply the affine transform to a frame as

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \Rightarrow \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Linear transformation

- 3-by-3 transform matrix → 4-by-4 affine transform

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & 0 \\ e & f & g & 0 \\ i & j & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}.$$

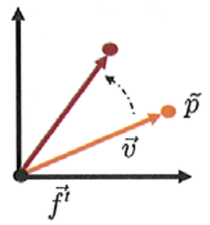
Linear transformation

- affine transformation:

$$L = \begin{bmatrix} l & 0 \\ 0 & 1 \end{bmatrix}$$

where, L is a 4-by-4 matrix; l is a 3-by-3 matrix.

- A linear transform is applied to a point. This is accomplished by applying the linear transform to its offset vector.



Translation transform

- translation transformation to points

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}.$$

Translation transform

- translation transformation to points

$$c_1 \Rightarrow c_1 + t_x$$
$$c_2 \Rightarrow c_2 + t_y$$
$$c_3 \Rightarrow c_3 + t_z$$

- translation matrix

$$T = \begin{bmatrix} i & t \\ 0 & 1 \end{bmatrix}$$

- where, T is a 4-by-4 matrix; i is a 3-by-3 identity matrix, t is 3-by-1 matrix for translation.

Affine transform matrix

- An affine matrix can be factored into a linear part and a translational part:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c & 0 \\ e & f & g & 0 \\ i & j & k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} l & t \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} i & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} l & 0 \\ 0 & 1 \end{bmatrix} \qquad A = TL$$

Affine transform matrix

- NB as matrix multiplication is not commutative, the order of the multiplication *TL* matters!!!

$$TL \neq LT$$

- Since these matrices have the same size (4-by-4), it is difficult to debug when you messed up the order. Pay extra attention on it while you are coding…

Rigid body transformation

- When the linear transform is a rotation, we call this as rigid body transformation (rotation + translation only).

$$A = TR$$

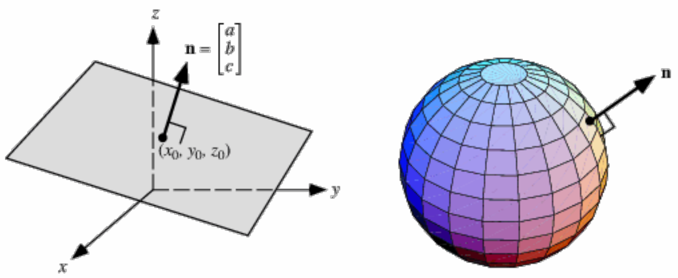
- A rigid body transformation preserves dot product between vectors, handedness of a basis, and distance between points.
- Its geometric topology is maintained while transforming it.

Affine transform acting on vector

- If fourth coordinate of **c** is zero, this just transforms a vector to a vector.
  - note that the fourth column is irrelevant
  - a vector cannot be translated

Normals

- Normal: a vector that is orthogonal to the tangent plane of the surfaces at that point.
  - the tangent plane is a plane of vectors that are defined by subtracting (infinitesimally) nearby surface points: $\vec{n} \cdot (\tilde{p}_1 - \tilde{p}_2) = 0$

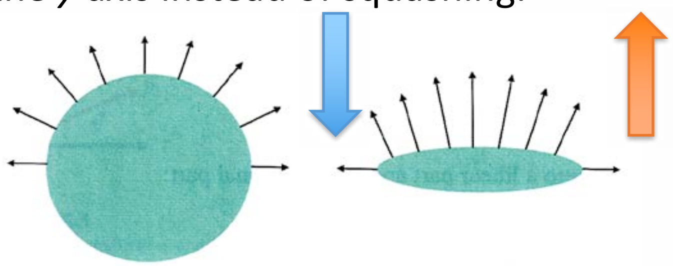


Normals

- We use normals for shading
- how do they transform
- suppose *i* rotate forward
  - normal gets rotated forward
- suppose squash in the *y* direction

Changing a shape

- Squashing a sphere makes its normals stretch along the *y* axis instead of squashing.



- normal gets higher in the y direction
- what is the rule?

$$\begin{bmatrix} nx \\ ny \\ nz \end{bmatrix} \neq \begin{bmatrix} nx' \\ ny' \\ nz' \end{bmatrix}.$$

Transforming normals

- Since the normal $\vec{n}$ and very close points $\tilde{p}_1$ and $\tilde{p}_2$ are on a surface: $\vec{n}\cdot(\tilde{p}_1-\tilde{p}_2)=0$

$$\begin{bmatrix} nx & ny & nz & * \end{bmatrix}\left(\begin{bmatrix} x1 \\ y1 \\ z1 \\ 1 \end{bmatrix}-\begin{bmatrix} x0 \\ y0 \\ z0 \\ 1 \end{bmatrix}\right)=0\,.$$

- After applying an affine transform *A*,

the normal of the transformed geometry

$$\left(\begin{bmatrix} nx & ny & nz & * \end{bmatrix}A^{-1}\right)\left(A\left(\begin{bmatrix} x1 \\ y1 \\ z1 \\ 1 \end{bmatrix}-\begin{bmatrix} x0 \\ y0 \\ z0 \\ 1 \end{bmatrix}\right)\right)=0\,.$$

Transforming normals

- Transformed normals:

$$\begin{bmatrix} nx' & ny' & nz' \end{bmatrix}=\begin{bmatrix} nx & ny & nz \end{bmatrix}l^{-1}\,.$$

- Transposing this expression:

$$\begin{bmatrix} nx' \\ ny' \\ nz' \end{bmatrix}=l^{-t}\begin{bmatrix} nx \\ ny \\ nz \end{bmatrix}.$$

Transforming normals

- Remember *l* is a rotation matrix (orthonormal), thus its inverse transpose is the same as the original: $l^{-t} = l$.

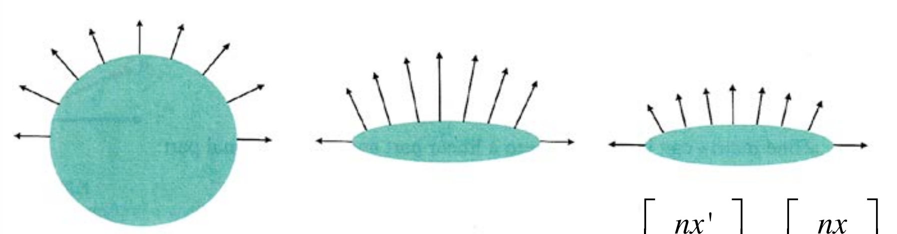
$$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$LL^t = I\ (L^t = L^{-1}), \det L = 1$$

- inverse transpose
  - so inverse transpose/transpose inverse is the rule
  - for rotation, transpose = inverse
  - for scale, transpose = nothing
  - in the code next week, we will send *A* and l^{-t} to the vertex shader.

Transforming normals

- Renormalize to correct unit normals of squashed shape:



$$l^{-1}\begin{bmatrix} nx' \\ ny' \\ nz' \end{bmatrix} \Rightarrow \begin{bmatrix} nx \\ ny \\ nz \end{bmatrix}.$$