3D transformations

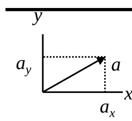
Uses of transformations

- **Modeling: position and resize parts of a complex model;**
- **Viewing: define and position the virtual camera**
- **Animation: define how objects move/change with time**

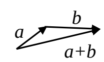
Types of transformations

- **rigid (isometric): preserve distances**
  - **rotations + translations**
- **similarity: distances are uniformly scaled, preserve angles**
  - **+ uniform scale**
- **affine: preserve parallel lines**
  - **+ skew**
- **perspective: preserve straight lines**

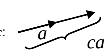
Review of vectors



Addition:



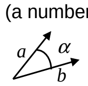
Multiplication by a number c:



Scalar (dot) product (a number):

2D: $(a,b) = a_x b_x + a_y b_y$

3D: $(a,b) = a_x b_x + a_y b_y + a_z b_z$



$(a,b) = |a||b|\cos\alpha$ $(a,b) = (b,a)$

$(a, k_1 b + k_2 c) = k_1 (a,b) + k_2 (a,c)$

Cross product (a vector):

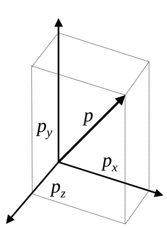
3D: $a \times b = \det \begin{pmatrix} \mathbf{e_x} & \mathbf{e_y} & \mathbf{e_z} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix}$

$|a \times b| = |a||b| \sin\alpha$

$a \times b = -b \times a$

$a \times (k_1 b + k_2 c) = k_1 a \times b + k_2 a \times c$

Vectors in 3D



$$p = p_x e_x + p_y e_y + p_z e_z$$

unit basis vectors

Homogeneous coordinates

regular 3D point to homogeneous:

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \longrightarrow \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

homogeneous point to regular 3D:

$$\begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \longrightarrow \begin{pmatrix} p_x/p_w \\ p_y/p_w \\ p_z/p_w \end{pmatrix}$$

Points vs. vectors

- a translated point is a different point
- a translated vector is the same vector
- in homogeneous coordinates:
  - points have last component nonzero
  - vectors have last component zero

Translation and scaling

Similar to 2D; translation by a vector

$$t = [t_x, t_y, t_z] \qquad \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Nonuniform scaling in three directions

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

Rotations around coord axes

angle θ, around X axis:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

around Y axis:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

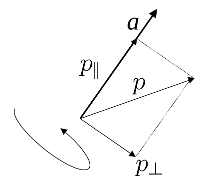
note where the minus is!

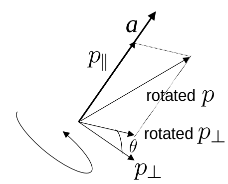
around Z axis:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

General rotations

Given an axis (a unit vector) and an angle, find the matrix





Only the component perpendicular to axis changes

General rotations

(rotated vectors are denoted with $'$)

project p on *a:* $\quad p_{\|} = (p, a)a$

the rest of *p* is the other component: $\quad p_{\perp} = p - (p, a)a$

rotate perp. component: $\quad p'_{\perp} = p_{\perp} \cos\theta + (a \times p_{\perp}) \sin\theta$

add back two components: $\quad p' = p'_{\perp} + p_{\|}$

Combine everything, using $a \times p_{\perp} = a \times p$ to simplify:

$$\boxed{p' = (p - (p, a)a) \cos\theta + (a \times p) \sin\theta + (p, a)a}$$

General rotations

How do we write all this using matrices?

$$p' = \cos\theta \; p + (1 - \cos\theta)(p, a)a + \sin\theta(a \times p)$$

$$(p, a)a = \begin{bmatrix} a_x a_x p_x + a_x a_y p_y + a_x a_z p_z \\ a_y a_x p_x + a_y a_y p_y + a_y a_z p_z \\ a_z a_x p_x + a_z a_y p_y + a_z a_z p_z \end{bmatrix} = \begin{bmatrix} a_x a_x & a_x a_y & a_x a_z \\ a_y a_x & a_y a_y & a_y a_z \\ a_z a_x & a_z a_y & a_z a_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

$$(a \times p) = \begin{bmatrix} -a_z p_y + a_y p_z \\ a_z p_x - a_x p_z \\ -a_y p_x + a_x p_y \end{bmatrix} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Final result, the matrix for a general rotation around a by angle θ:

$$\cos\theta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1 - \cos\theta) \begin{bmatrix} a_x a_x & a_x a_y & a_x a_z \\ a_y a_x & a_y a_y & a_y a_z \\ a_z a_x & a_z a_y & a_z a_z \end{bmatrix} + \sin\theta \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

Transformation of normals

- Normals are different from regular vectors
- a normal is defined as perpendicular to a line (2D) or a surface (3D)
- to transform a normal, transform the plane perpendicular to the normal

Transformation of normals

The normal to a plane spanned by v_1 and v_2 is defined by two equations up to a scalar:

$$(n, v_1) = 0,\ (n, v_2) = 0$$

The transformed normal should satisfy

$$(n', Mv_1) = 0,\ (n', Mv_2) = 0$$

Rewriting, we get:

$$(n', Mv_1) = (n')^T M v_1 = (M^T n')^T v_1 = 0$$
$$(n', Mv_2) = (n')^T M v_2 = (M^T n')^T v_2 = 0$$

This means $M^T n'$ is parallel to n ; we can take it to be equal to n, that is $n' = (M^T)^{-1} n$ (unnormalized)

Transformation of normals

Composition of transformations

- Recal that order matters (rotation * translation ≠ translation * rotation)
- Two ways to think about the order of transformations of an object
  - in a global coordinate system
  - in a coordinate system attached to the object
  - stick to a single method

Transformation order

- When we write transformations using standard math notation, the closest transformation to the point is applied first:

$$R\,T\,S\,p = R(T(Sp))$$

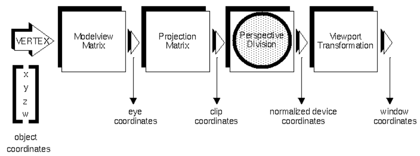
- first, the point is scaled,
  then translated,
  then rotated

OpenGL code order

- OpenGL maintains the current transformation;
- Each OpenGL function call that defines a transformation, postmultiplies the current matrix:
- All functions setting transformations should be before the object is drawn
- The order of function calls corresponds to the order of matrices in math notation:

```
glRotate(theta,ax,ay,az);
glTranslatef(tx,ty,tz);
glScalef(sx,sy,sz);
glBegin(GL_POINTS);
glVertex3f(px,py,pz);
glEnd();
```

Graphics pipeline



Modelview: model (position objects) + view (position the camera)

Projection: map viewing volume to a standard cube

Perspective divisiion: project 3D to 2D

Viewport: map the square [-1,1]x[-1,1]
in normalized device coordinates to the screen

OpenGL matrices

- **In OpenGL, all vectors (at least internally) have 4 components and all transformations are represented by 4 by 4 matrices**
- **the transformation matrices are stored as sequences of 16 numbers in the following order:**

$$\begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Note: the indices in the matrix above ***are not C array indices;*** they start with 1. Never use two-dimensional C arrays to represent OpenGL matrices, use one-dimensional arrays of length 16.