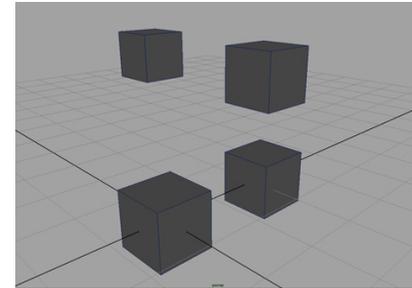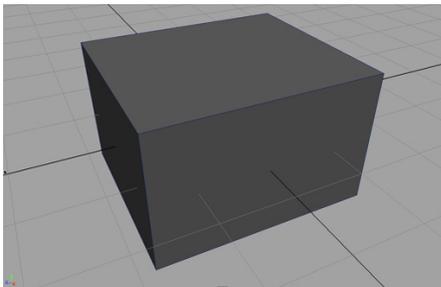# 3D Transformations

CS 4620 Lecture 8

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
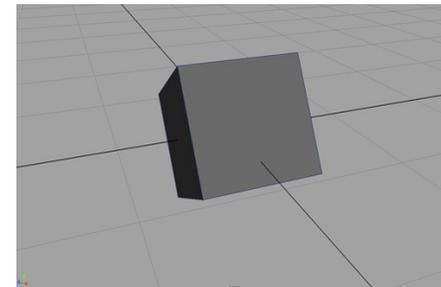
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
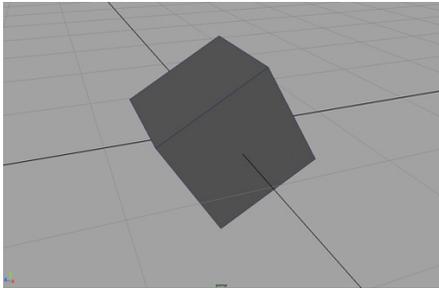
Rotation about z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
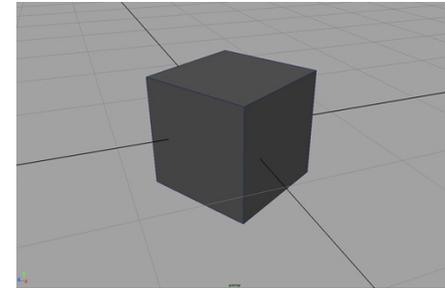
Rotation about *x* axis

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

Rotation about y axis

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

Transformations in OpenGL

- Stack-based manipulation of model-view transformation, M
- **glMatrixMode(GL_MODELVIEW)** Specifies model-view matrix
- **glLoadIdentity()** $M \leftarrow$ 4x4 identity
- **glTranslatef(float ux, float uy, float uz)** $M \leftarrow MT$
- **glRotatef(float theta, float ux, float uy, float uz)** $M \leftarrow MR$
- **glScalef(float sx, float sy, float sz)** $M \leftarrow MS$
- **glLoadMatrixf(float[] A)** $M \leftarrow A$ (Note: column major)
- **glMultMatrixf(float[] A)** $M \leftarrow MA$ (Note: column major)
- Manipulate matrix stack using:
  - **glPushMatrix()**
  - **glPopMatrix()**

Transformations in OpenGL

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
{// Draw something:
   glPushMatrix();
   glTranslatef(...);
   glRotatef(15f, ...);
   {// set color and draw simplices
      glBegin(GL_TRIANGLES);
      glColor3f(...);
      glVertex3f(...);
      glVertex3f(...);
      glVertex3f(...);
      glEnd();
   }
   glPopMatrix(); // toss old transform
}
{// Draw something else:
   glPushMatrix();
   ...
   glPopMatrix(); // toss old transform
}
```

General Rotation Matrices

- A rotation in 2D is around a point

- A rotation in 3D is around an axis
  - so 3D rotation is w.r.t a line, not just a point
  - there are many more 3D rotations than 2D
    - a 3D space around a given point, not just 1D

2D   3D
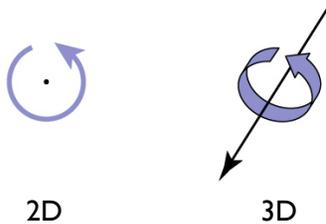
Specifying rotations

- In 2D, a rotation just has an angle
  - if it's about a particular center, it's a point and angle

- In 3D, specifying a rotation is more complex
  - basic rotation about origin: unit vector (axis) and angle
    - convention: positive rotation is CCW when vector is pointing at you
  - about different center: point (center), unit vector, and angle
    - this is redundant: think of a second point on the same axis...

- Alternative: Euler angles
  - stack up three coord axis rotations
  - degeneracies exist, e.g., gimbal lock

Coming up with the matrix

- Showed matrices for coordinate axis rotations
  - but what if we want rotation about some random axis?

- Compute by composing elementary transforms
  - transform rotation axis to align with x axis
  - apply rotation
  - inverse transform back into position

- Just as in 2D this can be interpreted as a similarity transform

Building general rotations

- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about y to get into x-y plane
  - rotate about z to align with x axis

- Alternative: construct frame and change coordinates
  - choose p, u, v, w to be orthonormal frame with p and u matching the rotation axis
  - apply similarity transform $T = F\,R_x(\theta)\,F^{-1}$

Orthonormal frames in 3D

- Useful tools for constructing transformations

- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right-handed ONB
    - that is, an **o**rtho**n**ormal **b**asis

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
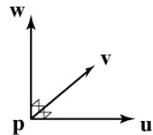
Building 3D frames

- Given a vector **a** and a secondary vector **b**
  - The **u** axis should be parallel to **a**; the **u**–**v** plane should contain **b**
    - $\mathbf{u} = \mathbf{u}$ / ||**u**||
    - $\mathbf{w} = \mathbf{u} \times \mathbf{b}$; $\mathbf{w} = \mathbf{w}$ / ||w||
    - $\mathbf{v} = \mathbf{w} \times \mathbf{u}$

- Given just a vector **a**
  - The **u** axis should be parallel to **a**; don't care about orientation about that axis
    - Same process but choose arbitrary **b** first
    - Good choice is not near **a**: e.g. set smallest entry to 1


Building general rotations

- Alternative: construct frame and change coordinates
  - choose p, u, v, w to be orthonormal frame with p and u matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$
  - interpretation: move to x axis, rotate, move back
  - interpretation: rewrite u-axis rotation in new coordinates
  - (each is equally valid)

- Or just derive the formula once, and reuse it (more later)

Building transforms from points

- Recall 2D affine transformation has 6 degrees of freedom (DOFs)
  - this is the number of "knobs" we have to set to define one

- Therefore 6 constraints suffice to define the transformation
  - handy kind of constraint: point **p** maps to point **q** (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs (i.e. can map any triangle to any other triangle)

- 3D affine transformation has 12 degrees of freedom
  - count them by looking at the matrix entries we're allowed to change

- Therefore 12 constraints suffice to define the transformation
  - in 3D, this is 4 point constraints (i.e. can map any tetrahedron to any other tetrahedron)

Transforming normal vectors

- Transforming surface normals
  - differences of points (and therefore tangents) transform OK
  - normals do not
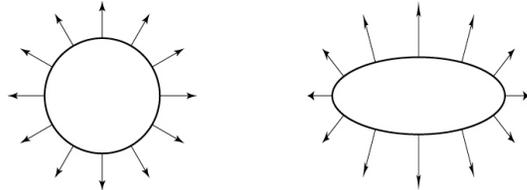
have: $\mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$

want: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0$

so set $X = (M^T)^{-1}$

then: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$

Derivation of General Rotation Matrix

- General 3x3 3D rotation matrix

- General 4x4 rotation about an arbitrary point