CS 4731: Computer Graphics
Lecture 8: 3D Affine transforms

Emmanuel Agu

Introduction to Transformations

- Introduce 3D affine transformation:
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shapes (shear)
- Previously developed 2D (x,y)
- Now, extend to 3D or (x,y,z) case
- Extend transform matrices to 3D
- Enable transformation of points by multiplication

Point Representation

- Previously, point in 2D as column matrix

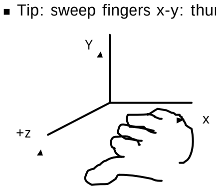
$$\begin{pmatrix} x \\ y \end{pmatrix} \Rightarrow \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Now, extending to 3D, add z-component:

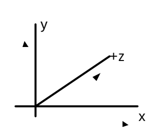
$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \quad \text{or} \quad P = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Coordinate Systems

- All perpendicular: *X* x *Y* = *Z* ; *Y* x *Z* = *X*; *Z* x *X* = *Y*;
- Tip: sweep fingers x-y: thumb is z



Right hand coordinate system



Left hand coordinate system
•Not used in this class and
•Not in OpenGL

Transforms in 3D

- 2D: 3x3 matrix multiplication
- 3D: 4x4 matrix multiplication: homogenous coordinates
- Recall: transform object = transform each vertice
- General form:

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix} \xrightarrow{\text{Xform of } P} \begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3x3 2D Translation Matrix

- Previously, 2D :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

3x3 2D Translation Matrix

- Now, 3D :

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}$$

OpenGL:
gltranslated(tx,ty,tz)

$$\Downarrow$$

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

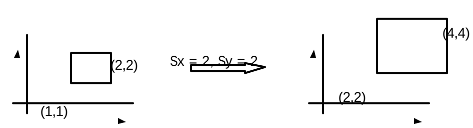
- Where: *x'= x.1 + y.0 + z.0 + tx.1 = x + tx*, … etc

2D Scaling

- Scale: Alter object size by scaling factor (s_x, s_y). i.e

x' = x . Sx
y' = y . Sy

$$\Rightarrow \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



3x3 2D Scaling Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

⬇

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

4x4 3D Scaling Matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

⬇

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

- Example:
- If *Sx* = *Sy* = *Sz* = 0.5
- Can scale:
- big cube (sides = 1) to small cube (sides = 0.5)
- 2D: square, 3D cube

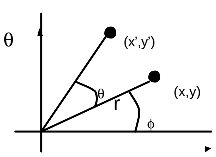
OpenGL:

glScaled(Sx,Sy,Sz)

Rotation

(x,y) -> Rotate *about the origin* by θ

→ (x', y')



How to compute (x', y') ?

x = r cos (f) y = r sin (f)

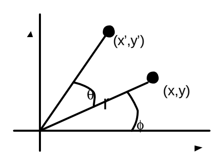
x' = r cos (f + q) y = r sin (f + q)

Rotation

Using trig identity

x' = x cos(q) – y sin(q)

y' = y cos(q) + x sin(q)



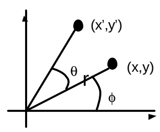
Matrix form?

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

3 x 3?

3x3 2D Rotation Matrix

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(q) & -\sin(q) \\ \sin(q) & \cos(q) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



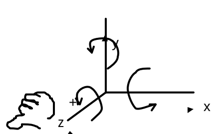
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(q) & -\sin(q) & 0 \\ \sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Rotating in 3D

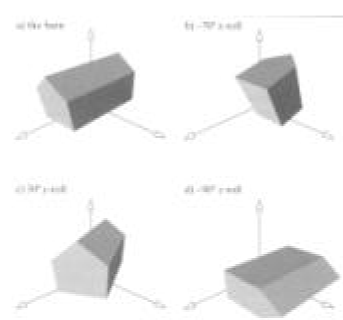
- Cannot do mindless conversion like before
- Why?
  - Rotate about what axis?
  - 3D rotation: about a defined axis
  - Different Xform matrix for:
    - Rotation about x-axis
    - Rotation about y-axis
    - Rotation about z-axis
- New terminology
  - X-roll: rotation about x-axis
  - Y-roll: rotation about y-axis
  - Z-roll: rotation about z-axis

Rotating in 3D

- New terminology
  - X-roll: rotation about x-axis
  - Y-roll: rotation about y-axis
  - Z-roll: rotation about z-axis
- Which way is +ve rotation
  - Look in –ve direction (into +ve arrow)
  - CCW is +ve rotation



Rotating in 3D



Rotating in 3D

- For a rotation angle, *b* about an axis
- Define:

$$c = \cos(b) \qquad s = \sin(b)$$

A x-roll:

$$R_x(b) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c & -s & 0 \\ 0 & s & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL:

glrotated(*q*, *1*, *0*, *0*)

Rotating in 3D

A y-roll:

$$R_y(b) = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL:

glrotated(*q*, *0*, *1*, *0*)

Rules:

- Rotate row, column int. is 1
- Rest of row/col is 0
- c,s in rect pattern

A z-roll:

$$R_z(b) = \begin{pmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

OpenGL:

glrotated(*q*, *0*, *0*, *1*)

Rotating in 3D

Q: Using y-roll equation, rotate *P* = (3,1,4) by 30 degrees:

A: c = cos(30) = 0.866, s = sin(30) = 0.5, and

$$Q = \begin{pmatrix} c & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ -s & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 4.6 \\ 1 \\ 1.964 \\ 1 \end{pmatrix}$$

E.g. first line: 3.c + 1.0 + 4.s + 1.0 = 4.6

Rotating in 3D

Q: Write C code to Multiply point P = (Px, Py, Pz, 1) by a 4x4 matrix shown below to give new point Q = (Qx,Qy,Qz, 1). i.e.

$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = M \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

where

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotating in 3D

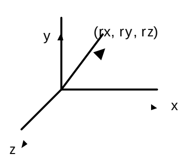
- Outline of solution:
  - Declare *P*,*Q* as array:
    - Double P[4], Q[4];
  - Declare transform matrix as 2-dimensional array
    - Double M[4][4];
  - Remember: C indexes from 0, not 1
  - Long way:
    - Write out equations line by line expression for Q[i]
    - E.g. Q[0] = P[0]*M[0][0] + P[1]*M[0][1] + P[2]*M[0][2] + P[3]*M[0][3]
  - Cute way:
    - Use indexing, say i for outer loop, j for inner loop

Rotating in 3D

- Using loops looks like:
  - ```
    for(i=0;i<4;i++)
    {
      temp = 0;
      for(j=0;j<4;j++)
      {
          temp += P[j]*M[i][j];
      }
      Q[i] = temp;
    }
    ```
- Test matrice code rigorously
- Use known results (or by hand) and plug into your code

3D Rotation About Arbitrary Axis

- Arbitrary rotation axis (rx, ry, rz)
- openGL: rotate(θ, rx, ry, rz)
- Without openGL: a little hairy!!
- Important: read Hill pp. 239-241



3D Rotation About Arbitrary Axis

- Can compose arbitrary rotation as combination of:
  - X-roll
  - Y-roll
  - Z-roll

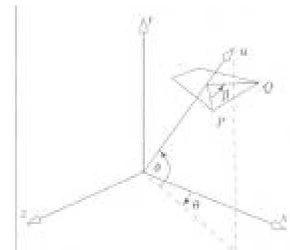
$$M = R_z(b_3)R_y(b_2)R_x(b_1)$$
    ```

3D Rotation About Arbitrary Axis

- Classic: use Euler's theorem
- Euler's theorem: any sequence of rotations = one rotation about some axis
- Our approach:
  - Want to rotate β about the axis **u** through origin and arbitrary point
  - Use two rotations to align **u** and x-axis
  - Do x-roll through angle β
  - Negate two previous rotations to de-align **u** and x-axis

3D Rotation About Arbitrary Axis

$$R_u(b) = R_y(-q)R_z(f)R_x(b)R_z(-f)R_y(q)$$



Composing Transformation

- Composing transformation – applying several transforms in succession to form one overall transformation
- Example:

  M1 X M2 X M3 X P

  where M1, M2, M3 are transform matrices applied to P
- Be careful with the order
- Matrix multiplication is not commutative

References

- Hill, chapter 5.3