Transformations

Week 2, Wed Jan 12
http://www.ugrad.cs.ubc.ca/~cs314/Vjan2005

Reading
FCG Chap 5 (except 5.1.6, 5.3.1),
FCG pages 224-225
RB Chap Viewing:
  * Sect. Viewing and Modeling Transforms until Viewing Transformations
  * Sect. Examples of Composing Several Transformations through Building an Articulated Robot Arm
RB Appendix Homogeneous Coordinates and Transformation Matrices
  * until Perspective Projection

Textbook Errata
  * list at http://www.cs.utah.edu/~shirley/fcg/errata
    * math review: also p 48
    * a x (b x c) != (a x b) x c
    * transforms: p 91
    * should halve x (not y) in Fig 5.10
    * transforms: p 106
    * second line of matrices: \([x_p, y_p, 1]\)

Review: Rendering Pipeline

Review: OpenGL
  * pipeline processing, set state as needed
    void display()
    {
      glClearColor(0.0, 0.0, 0.0, 0.0);
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(0.0, 1.0, 0.0);
      glBegin(GL_POLYGON);
      glVertex3f(0.25, 0.25, -0.5);
      glVertex3f(0.75, 0.25, -0.5);
      glVertex3f(0.75, 0.75, -0.5);
      glVertex3f(0.25, 0.75, -0.5);
      glEnd();
      glFlush();
    }

Review: Event-Driven Programming
  * main loop not under your control
    * vs. procedural
  * control flow through event callbacks
    * redraw the window now
    * key was pressed
    * mouse moved
  * callback functions called from main loop when events occur
    * mouse/keyboard state setting vs. redrawing
Keyboard/Mouse Callbacks
- do minimal work
- request redraw for display
- example: keypress triggering animation
  - do not create loop in input callback!
    - what if user hits another key during animation?
- shared/global variables to keep track of state
- display function acts on current variable value

Transformations
- transforming an object = transforming all its points
- transforming a polygon = transforming its vertices

Matrix-Vector Multiplication
\[
\begin{bmatrix}
M_{11} & M_{12} & M_{13} \\
M_{21} & M_{22} & M_{23} \\
M_{31} & M_{32} & M_{33}
\end{bmatrix}
\begin{bmatrix}
v_1 \\
v_2 \\
v_3
\end{bmatrix} =
\begin{bmatrix}
u \\
v \\
v
\end{bmatrix}
\]
- linear transformation of a vector
  - maps one vector to another
  - preserves linear combinations
- any linear transform can be represented by a matrix
  - scaling
  - rotation
  - translation

Scaling
- **scaling** a coordinate means multiplying each of its components by a scalar
- **uniform scaling** means this scalar is the same for all components:
- **non-uniform scaling**: different scalars per component:
  - how can we represent this in matrix form?
Scaling

- scaling operation:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} = \begin{bmatrix}
  ax \\
  by
  \end{bmatrix}
  \]

- or, in matrix form:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} = \begin{bmatrix}
  a & 0 \\
  0 & b
  \end{bmatrix} \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

2D Rotation

\[
\begin{align*}
  x' &= x \cos(\theta) - y \sin(\theta) \\
  y' &= x \sin(\theta) + y \cos(\theta)
\end{align*}
\]

2D Rotation From Trig Identities

- \( x = r \cos(\phi) \)
- \( y = r \sin(\phi) \)
- \( x' = r \cos(\phi + \theta) \)
- \( y' = r \sin(\phi + \theta) \)

Trig Identity...

- \( x' = r \cos(\phi) \cos(\theta) - r \sin(\phi) \sin(\theta) \)
- \( y' = r \sin(\phi) \cos(\theta) + r \cos(\phi) \sin(\theta) \)

Substitute...

- \( x' = x \cos(\theta) - y \sin(\theta) \)
- \( y' = x \sin(\theta) + y \cos(\theta) \)

2D Rotation Matrix

- easy to capture in matrix form:
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix} = \begin{bmatrix}
  \cos(\theta) & -\sin(\theta) \\
  \sin(\theta) & \cos(\theta)
  \end{bmatrix} \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]

- even though \( \sin(\theta) \) and \( \cos(\theta) \) are nonlinear functions of \( \theta \),
- \( x' \) is a linear combination of \( x \) and \( y \)
- \( y' \) is a linear combination of \( x \) and \( y \)

2D Rotation: Another Derivation

- \( x' = x \cos(\theta) - y \sin(\theta) \)
- \( y' = x \sin(\theta) + y \cos(\theta) \)

2D Rotation: Another Derivation

- \( x' = x \cos(\theta) - y \sin(\theta) \)
- \( y' = x \sin(\theta) + y \cos(\theta) \)
2D Rotation: Another Derivation

\[
(x', y') = \begin{pmatrix} x \\ y \end{pmatrix}' = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \begin{pmatrix} x + a \\ y + b \end{pmatrix} = \begin{pmatrix} x' \\ y' \end{pmatrix}
\]

scaling matrix

rotation matrix

2D Translation

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x & 0 \\ 0 & y \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}
\sin(\theta) \cos(\theta) \begin{pmatrix} x \\ y \end{pmatrix}
\]

- \[
x' = x \cos(\theta) - y \sin(\theta)
\]

- \[
y' = x \sin(\theta) + y \cos(\theta)
\]

- \[
x' = A - B
\]

- \[
A = x \cos(\theta)
\]

- \[
B = y \sin(\theta)
\]
Homogeneous Coordinates Geometrically

- **homogeneous**
  - (x, y, w)
- **cartesian**
  - \(\left(\frac{x}{w}, \frac{y}{w}\right)\)

- point in 2D cartesian + weight w = point P in 3D homog. coords
- multiple of (x,y,w)
- form a line L in 3D
- all homogeneous points on L represent same 2D cartesian point
- example: (2,2,1) = (4,4,2) = (1,1,0.5)

- homogenize to convert homog. 3D point to cartesian 2D point:
  - divide by w to get (x/w, y/w, 1)
  - projects line to point onto w=1 plane
  - when w=0, consider it as direction
  - points at infinity
  - these points cannot be homogenized
  - (0,0,0) is undefined
Homogeneous Coordinates Geometrically

- \( w = 0 \) denotes points at infinity
- think of as direction
- cannot be homogenized
- lies on x-y plane
- \((0,0,0)\) is not allowed

Homogeneous Coordinates Summary

- may seem unintuitive, but they make graphics operations much easier
- allow all linear transformations to be expressed through matrix multiplication
- use 4x4 matrices for 3D transformations

3D Rotation About Z Axis

\[
\begin{align*}
x' &= x \cos \theta - y \sin \theta \\
y' &= x \sin \theta + y \cos \theta \\
z' &= z
\end{align*}
\]

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

- general OpenGL command
- \( \text{glRotatef}(\text{angle},0,0,1); \)

3D Rotation in X, Y

- around x axis: \( \text{glRotatef}(\text{angle},1,0,0); \)
- \[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

- around y axis: \( \text{glRotatef}(\text{angle},0,1,0); \)
- \[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

3D Scaling

\[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} =
\begin{bmatrix}
a & 0 & 0 & 0 \\
0 & b & 0 & 0 \\
0 & 0 & c & 0 \\
1 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

\( \text{glScalef}(a,b,c); \)

3D Translation

\[
\begin{bmatrix}
x' \\
y' \\
z'
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 & a \\
0 & 1 & 0 & b \\
0 & 0 & 1 & c \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

\( \text{glTranslatef}(a,b,c); \)