Transformations

January 28, 2013
Jernej Barbic
University of Southern California

http://www-bcf.usc.edu/~jbarbic/cs480-s13/
OpenGL Transformations
OpenGL Transformation Matrices

- Model-view matrix (4x4 matrix)
- Projection matrix (4x4 matrix)
4x4 Model-view Matrix (this lecture)

- Translate, rotate, scale objects
- Position the camera

vertices in canonical 3D world coordinate system

vertices in 3D

Model-view

Projection

vertices in 2D
4x4 Projection Matrix (next lecture)

- Project from 3D to 2D

vertices in 3D → Model-view → Projection → vertices in 2D

vertices in canonical 3D world coordinate system
OpenGL Transformation Matrices

- Manipulated separately in OpenGL (must set matrix mode):
  
  ```
  glMatrixMode (GL_MODELVIEW);
  glMatrixMode (GL_PROJECTION);
  ```
Setting the Current Model-view Matrix

• Load or post-multiply

```cpp
glMatrixMode (GL_MODELVIEW);
glLoadIdentity(); // very common usage
float m[16] = { … };  
glLoadMatrixf(m); // rare, advanced
glMultMatrixf(m); // rare, advanced
```

• Use library functions

```cpp
glTranslatef(dx, dy, dz);
glRotatef(angle, vx, vy, vz);
glScalef(sx, sy, sz);
```
Translated, rotated, scaled object
The *rendering* coordinate system

Initially (after `glLoadIdentity()`):

rendering coordinate system = world coordinate system
The *rendering* coordinate system

```c
glTranslatef(x, y, z);
```

[x, y, z]

world

rendering coordinate system
The rendering coordinate system

`glRotatef(angle, ax, ay, az);`
The *rendering* coordinate system

glScalef(sx, sy, sz);
OpenGL code

```c
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();
glTranslatef(x, y, z);
glRotatef(angle, ax, ay, az);
glScalef(sx, sy, sz);
renderBunny();
```
Rendering more objects

How to obtain this frame?

world
How to obtain this frame?

Solution 1:

Find `glTranslatef(...), glRotatef(...), glScalef(...)`
Solution 2: \texttt{gl\{Push,Pop\}Matrix}

\begin{verbatim}
glMatrixMode (GL_MODELVIEW);
glLoadIdentity();

// render first bunny
glPushMatrix(); // store current matrix
glTranslatef(...);
glRotatef(...);
renderBunny();
glPopMatrix(); // pop matrix

// render second bunny
glPushMatrix(); // store current matrix
glTranslatef(...);
glRotatef(...);
renderBunny();
glPopMatrix(); // pop matrix
\end{verbatim}
3D Math Review
 Scalars

• Scalars \( \alpha, \beta, \gamma \) from a **scalar field**
• Operations \( \alpha + \beta, \alpha \cdot \beta, 0, 1, -\alpha, (\ )^{-1} \)
• “Expected” laws apply
• Examples: rationals or reals with addition and multiplication
Vectors

- Vectors $u, v, w$ from a **vector space**
- Vector addition $u + v$, subtraction $u - v$
- Zero vector $0$
- Scalar multiplication $\alpha v$
Euclidean Space

• Vector space over real numbers

• Three-dimensional in computer graphics

• Dot product: $\alpha = u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3$

• $0 \cdot 0 = 0$

• $u, v$ are orthogonal if $u \cdot v = 0$

• $|v|^2 = v \cdot v$ defines $|v|$, the length of $v$
Lines and Line Segments

- Parametric form of line: $P(\alpha) = P_0 + \alpha \ d$

- Line segment between $Q$ and $R$:
  $P(\alpha) = (1-\alpha) \ Q + \alpha \ R \ \text{for} \ 0 \leq \alpha \leq 1$
Convex Hull

- Convex hull defined by

\[ P = \alpha_1 P_1 + \ldots + \alpha_n P_n \]

for \( \alpha_1 + \ldots + \alpha_n = 1 \)

and \( 0 \leq \alpha_i \leq 1, \ i = 1, \ldots, n \)
Projection

- Dot product projects one vector onto another vector

\[ u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 = |u| |v| \cos(\theta) \]

\[ \text{pr}_v u = (u \cdot v) v / |v|^2 \]
Cross Product

\[
\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{pmatrix}
\]

- \( |a \times b| = |a| \cdot |b| \cdot |\sin(\theta)| \)

- Cross product is perpendicular to both \( a \) and \( b \)

- Right-hand rule

Plane

- Plane defined by point $P_0$ and vectors $u$ and $v$

- $u$ and $v$ should not be parallel

- Parametric form:
  \[ T(\alpha, \beta) = P_0 + \alpha \ u + \beta \ v \]  
  ($\alpha$ and $\beta$ are scalars)

- $n = u \times v / |u \times v|$ is the normal

- $n \cdot (P - P_0) = 0$ if and only if $P$ lies in plane
Coordinate Systems

• Let \( v_1, v_2, v_3 \) be three linearly independent vectors in a 3-dimensional vector space.

• Can write any vector \( w \) as

\[
    w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3
\]

for some scalars \( \alpha_1, \alpha_2, \alpha_3 \).
Frames

- Frame = origin \( P_0 \) + coordinate system
- Any point \( P = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 \)
In Practice, Frames are Often Orthogonal
Change of Coordinate System

- Bases \{u_1, u_2, u_3\} and \{v_1, v_2, v_3\}
- Express basis vectors \(u_i\) in terms of \(v_j\)

\[
\begin{align*}
  u_1 &= \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3 \\
  u_2 &= \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3 \\
  u_3 &= \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3
\end{align*}
\]

- Represent in matrix form:

\[
\begin{bmatrix}
  u_1 \\
  u_2 \\
  u_3
\end{bmatrix} =
\begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3
\end{bmatrix}
\]

\[
M =
\begin{bmatrix}
  \gamma_{11} & \gamma_{12} & \gamma_{13} \\
  \gamma_{21} & \gamma_{22} & \gamma_{23} \\
  \gamma_{31} & \gamma_{32} & \gamma_{33}
\end{bmatrix}
\]
Representing 3D transformations (and model-view matrices)
Linear Transformations

• 3 x 3 matrices represent linear transformations
  \[ a = M b \]
• Can represent rotation, scaling, and reflection
• Cannot represent translation

\[
M = \begin{bmatrix}
\gamma_{11} & \gamma_{12} & \gamma_{13} \\
\gamma_{21} & \gamma_{22} & \gamma_{23} \\
\gamma_{31} & \gamma_{32} & \gamma_{33}
\end{bmatrix}
\]
In order to represent rotations, scales AND translations:
Homogeneous Coordinates

• Augment $[\alpha_1 \alpha_2 \alpha_3]^T$ by adding a fourth component (1):
  $$\mathbf{p} = [\alpha_1 \alpha_2 \alpha_3 1]^T$$

• Homogeneous property:
  $$\mathbf{p} = [\alpha_1 \alpha_2 \alpha_3 1]^T = [\beta \alpha_1 \beta \alpha_2 \beta \alpha_3 \beta]^T$$,
  for any scalar $\beta \neq 0$
Homogeneous coordinates are transformed by 4x4 matrices

$q = A \ p$
Affine Transformations (4x4 matrices)

- Translation
- Rotation
- Scaling
- Any composition of the above
- Later: projective (perspective) transformations
  - Also expressible as 4 x 4 matrices!
Translation

• \( q = p + d \) where \( d = [\alpha_x \, \alpha_y \, \alpha_z \, 0]^T \)

• \( p = [x \, y \, z \, 1]^T \)

• \( q = [x' \, y' \, z' \, 1]^T \)

• Express in matrix form \( q = T \, p \) and solve for \( T \)

\[
T = \begin{bmatrix}
1 & 0 & 0 & \alpha_x \\
0 & 1 & 0 & \alpha_y \\
0 & 0 & 1 & \alpha_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Scaling

- \( x' = \beta_x x \)
- \( y' = \beta_y y \)
- \( z' = \beta_z z \)
- Express as \( q = S p \) and solve for \( S \)

\[
S = \begin{bmatrix}
\beta_x & 0 & 0 & 0 & 0 \\
0 & \beta_y & 0 & 0 & 0 \\
0 & 0 & \beta_z & 0 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]
Rotation in 2 Dimensions

- Rotation by $\theta$ about the origin
- $x' = x \cos \theta - y \sin \theta$
- $y' = x \sin \theta + y \cos \theta$

- Express in matrix form:

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

- Note that the determinant is 1
Rotation in 3 Dimensions

• Orthogonal matrices:

\[ RR^T = R^T R = I \]
\[ \det(R) = 1 \]

• Affine transformation:

\[
A = \begin{bmatrix}
R_{11} & R_{12} & R_{13} & 0 \\
R_{21} & R_{22} & R_{23} & 0 \\
R_{31} & R_{32} & R_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Affine Matrices are Composed by Matrix Multiplication

- \( A = A_1 A_2 A_3 \)
- Applied from right to left
- \( A \ p = (A_1 A_2 A_3) \ p = A_1 (A_2 (A_3 \ p)) \)
- When calling `glTranslate3f`, `glRotatef`, or `glScalef`, OpenGL forms the corresponding 4x4 matrix, and multiplies the current modelview matrix with it.
Summary

• OpenGL Transformation Matrices
• Vector Spaces
• Frames
• Homogeneous Coordinates
• Transformation Matrices