Transformations

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

- Introduce standard transformations
  - Rotations
  - Translation
  - Scaling
  - Shear
- Derive homogeneous coordinate transformation matrices
- Learn to build arbitrary transformation matrices from simple transformations
A transformation maps points to other points and/or vectors to other vectors.

\[ \mathbf{Q} = \mathbf{T}(\mathbf{P}) \]

\[ \mathbf{v} = \mathbf{T}(\mathbf{u}) \]
Pipeline Implementation

T (from application program)

transformation

rasterizer

frame buffer

vertices

pixels
Translation

• Move (translate, displace) a point to a new location

• Displacement determined by a vector $\mathbf{d}$
  - Three degrees of freedom
  - $\mathbf{P'} = \mathbf{P} + \mathbf{d}$
Object Translation

Every point in object is displaced by same vector
Translation Using Representations

Using the homogeneous coordinate representation in some frame

\[ \mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T \]

\[ \mathbf{p}' = \begin{bmatrix} x' & y' & z' & 1 \end{bmatrix}^T \]

\[ \mathbf{d} = \begin{bmatrix} dx & dy & dz & 0 \end{bmatrix}^T \]

Hence \( \mathbf{p}' = \mathbf{p} + \mathbf{d} \) or

\[ x' = x + dx \]

\[ y' = y + dy \]

\[ z' = z + dz \]

Note that this expression is in four dimensions and expresses that point = vector + point
Translation Matrix

We can also express translation using a 4 x 4 matrix \( T \) in homogeneous coordinates \( p' = Tp \) where

\[
T = T(d_x, d_y, d_z) = \begin{bmatrix}
1 & 0 & 0 & d_x \\
0 & 1 & 0 & d_y \\
0 & 0 & 1 & d_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

This form is better for implementation because all affine transformations can be expressed this way and multiple transformations can be concatenated together.
Rotation (2D)

Consider rotation about the origin by $\theta$ degrees:
- radius stays the same, angle increases by $\theta$

\[
x' = x \cos \theta - y \sin \theta
\]
\[
y' = x \sin \theta + y \cos \theta
\]
Rotation about the z-axis

- Rotation about z axis in three dimensions leaves all points with the same z
  - Equivalent to rotation in two dimensions in planes of constant z
    \[ x' = x \cos \theta - y \sin \theta \]
    \[ y' = x \sin \theta + y \cos \theta \]
    \[ z' = z \]
  - or in homogeneous coordinates
    \[ p' = R_z(\theta)p \]
Rotation Matrix

\[
R = R_z(\theta) = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
Rotation about x and y axes

• Same argument as for rotation about z-axis
  - For rotation about x-axis, $x$ is unchanged
  - For rotation about y-axis, $y$ is unchanged

$$ R = R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 & 0 \\ 0 & \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} $$

$$ R = R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} $$
Scaling

Expand or contract along each axis (fixed point of origin)

\[ x' = s_x x \]
\[ y' = s_y y \]
\[ z' = s_z z \]

\[ p' = S p \]

\[ S = S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \]
Reflection

corresponds to negative scale factors

\( s_x = -1 \) \( s_y = 1 \)

\( s_x = -1 \) \( s_y = -1 \)

\( s_x = 1 \) \( s_y = -1 \)
Inverses

• Although we could compute inverse matrices by general formulas, we can use simple geometric observations
  - Translation: $T^{-1}(d_x, d_y, d_z) = T(-d_x, -d_y, -d_z)$
  - Rotation: $R^{-1}(\theta) = R(-\theta)$
    • Holds for any rotation matrix
    • Note that since $\cos(-\theta) = \cos(\theta)$ and $\sin(-\theta) = -\sin(\theta)$
      $R^{-1}(\theta) = R^T(\theta)$
  - Scaling: $S^{-1}(s_x, s_y, s_z) = S(1/s_x, 1/s_y, 1/s_z)$
Concatenation

• We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices.

• Because the same transformation is applied to many vertices, the cost of forming a matrix \( M = ABCD \) is not significant compared to the cost of computing \( Mp \) for many vertices \( p \).

• The difficult part is how to form a desired transformation from the specifications in the application.
Order of Transformations

• Note that matrix on the right is the first applied
• Mathematically, the following are equivalent
  \[ p' = ABCp = A(B(Cp)) \]
• Note many references use column matrices to present points. In terms of column matrices
  \[ p'^T = p^T C^T B^T A^T \]
A rotation by $\theta$ about an arbitrary axis can be decomposed into the concatenation of rotations about the $x$, $y$, and $z$ axes

$$ R(\theta) = R_z(\theta_z) R_y(\theta_y) R_x(\theta_x) $$

$\theta_x \theta_y \theta_z$ are called the Euler angles.

Note that rotations do not commute.

We can use rotations in another order but with different angles.
Rotation About a Fixed Point other than the Origin

Move fixed point to origin
Rotate
Move fixed point back

\[ M = T(p_f) \ R(\theta) \ T(-p_f) \]
Shear

- Helpful to add one more basic transformation
- Equivalent to pulling faces in opposite directions
Shear Matrix

Consider simple shear along $x$ axis

\[ x' = x + y \cot \theta \]
\[ y' = y \]
\[ z' = z \]

\[ H(\theta) = \begin{bmatrix}
1 & \cot \theta & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 
\end{bmatrix} \]
3D Transformations

- A vertex is transformed by $4 \times 4$ matrices
- All matrices are stored column-major in OpenGL
  - this is opposite of what “C” programmers expect
- Matrices are always post-multiplied
  - product of matrix and vector is $\mathbf{Mv}$

$$
\mathbf{M} = \begin{bmatrix}
  m_0 & m_4 & m_8 & m_{12} \\
  m_1 & m_5 & m_9 & m_{13} \\
  m_2 & m_6 & m_{10} & m_{14} \\
  m_3 & m_7 & m_{11} & m_{15}
\end{bmatrix} \quad \mathbf{v} = \begin{bmatrix}
  v_1 \\
  v_2 \\
  v_3 \\
  v_4
\end{bmatrix}$$
Affine Transformations

\[
\begin{bmatrix}
    x' \\
    y' \\
    z' \\
    1
\end{bmatrix}
= \begin{bmatrix}
    m_0 & m_4 & m_8 & m_{12} \\
    m_1 & m_5 & m_9 & m_{13} \\
    m_2 & m_6 & m_{10} & m_{14} \\
    0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    z \\
    1
\end{bmatrix}
\]

- Characteristic of many important transformations
  - Translation
  - Rotation
  - Scaling
  - Shear

- Line preserving
OpenGL Transformations

Prof. George Wolberg
Dept. of Computer Science
City College of New York
Objectives

• Learn how to carry out transformations in OpenGL
  - Rotation
  - Translation
  - Scaling

• Introduce QMatrix4x4 and QVector3D transformations
  - Model-view
  - Projection
Current Transformation Matrix (CTM)

- Conceptually there is a 4 x 4 homogeneous coordinate matrix, the current transformation matrix (CTM) that is part of the state and is applied to all vertices that pass down the pipeline.
- The CTM is defined in the user program and loaded into a transformation unit.
CTM operations

• The CTM can be altered either by loading a new CTM or by postmultiplication
  Load an identity matrix: \( C \leftarrow I \)
  Load an arbitrary matrix: \( C \leftarrow M \)

  Load a translation matrix: \( C \leftarrow T \)
  Load a rotation matrix: \( C \leftarrow R \)
  Load a scaling matrix: \( C \leftarrow S \)

  Postmultiply by an arbitrary matrix: \( C \leftarrow CM \)
  Postmultiply by a translation matrix: \( C \leftarrow CT \)
  Postmultiply by a rotation matrix: \( C \leftarrow CR \)
  Postmultiply by a scaling matrix: \( C \leftarrow CS \)
Rotation about a Fixed Point

Start with identity matrix: \( C \leftarrow I \)
Move fixed point to origin: \( C \leftarrow CT \)
Rotate: \( C \leftarrow CR \)
Move fixed point back: \( C \leftarrow CT^{-1} \)

Result: \( C = TR T^{-1} \) which is backwards.

This result is a consequence of doing postmultiplications. Let’s try again.
Reversing the Order

We want \( C = T^{-1} R T \) so we must do the operations in the following order

\[
\begin{align*}
C &\leftarrow I \\
C &\leftarrow CT^{-1} \\
C &\leftarrow CR \\
C &\leftarrow CT
\end{align*}
\]

Each operation corresponds to one function call in the program.

*The last operation specified is the first executed in the program!*
Rotation, Translation, Scaling

Create an identity matrix:

```cpp
QMatrix4x4 m;
m.setToIdentity();
```

Multiply on right by rotation matrix of \textbf{theta} in degrees where \((vx, vy, vz)\) define axis of rotation

```cpp
m.rotate(theta, QVector3D(vx, vy, vz));
```

Do same with translation and scaling:

```cpp
m.scale(sx, sy, sz);
m.translate(dx, dy, dz);
```
Example

• Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)

```cpp
QMatrix4x4 m;
m.setToIdentity();
m.translate(1.0, 2.0, 3.0);
m.rotate(30.0, QVector3D(0.0, 0.0, 1.0));
m.translate(-1.0, -2.0, -3.0);
```

• Remember that the last matrix specified is the first applied
Arbitrary Matrices

• Can load and multiply by matrices defined in the application program
• Matrices are stored as one dimensional array of 16 elements which are the components of the desired $4 \times 4$ matrix stored by columns
• OpenGL functions that have matrices as parameters allow the application to send the matrix or its transpose
Vertex Shader for Rotation of Cube (1)

in vec4 vPosition;
in vec4 vColor;
out vec4 color;
uniform vec3 theta;

void main()
{
    // Compute the sines and cosines of theta for
    // each of the three axes in one computation.
    vec3 angles = radians( theta );
    vec3 c = cos( angles );
    vec3 s = sin( angles );
// Remember: these matrices are column-major

mat4 rx = mat4( 1.0, 0.0, 0.0, 0.0,
                0.0, c.x, s.x, 0.0,
                0.0, -s.x, c.x, 0.0,
                0.0, 0.0, 0.0, 1.0);

mat4 ry = mat4( c.y, 0.0, -s.y, 0.0,
                0.0, 1.0, 0.0, 0.0,
                s.y, 0.0, c.y, 0.0,
                0.0, 0.0, 0.0, 1.0);
mat4 rz = mat4( c.z, -s.z, 0.0, 0.0,
              s.z,  c.z, 0.0, 0.0,
              0.0,  0.0, 1.0, 0.0,
              0.0,  0.0, 0.0, 1.0 );

color = vColor;

gl_Position = rz * ry * rx * vPosition;

}
Sending Angles from Application

GLuint thetaID;  // theta uniform location
vec3 theta;      // axis angles

void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glUniform3fv( thetaID, 1, theta );
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
}