Composite Transformations

- Transforms may be done in sequence, hence they can be concatenated.
- This is a basis for
  - pipeline processing.
  - hierarchical CG data structures (scene graphs).

Example:
- Rotate an object about a point along the z axis:
  - Translate Object such that point is on the origin.
  - Rotate Object around the z axis.
  - Translate Object such that the point is back to its original location.

\[ p' = (A^{-1}(B(Ap))) \Rightarrow p' = A^{-1}BAp \]
\[ M = A^{-1}BA, \ p' = Mp \]
\[ p' = A^{-1}BAp \neq ABA^{-1}p \]
Composite Transformations…

• …are the basis for scene graphs
• main node types:
  • group (G)
  • transform T_x (and group)
• Concatenation is performed during traversals
• Transforms represented as 4x4 matrices will be multiplied
• Geometry sent down the pipe will undergo combined transformation
• For Geo_1 to be w.r.t. the CS as defined by T1 (and hence G):

\[ \forall p \in Geo_1 : T_1 \cdot T_2 \cdot T_3 \cdot p \]

Composite Transformations

• Composite transformations using matrices is supported
  • by floating point units in the graphics hardware.
  • by low-level matrix stacks which enable hierarchies and scene graph traversal.

OpenGL example

```c
GLfloat M0, M1, M2, M3, M4;

void render(){
    M0 = loadMatrix();
    M1 = pushMatrix();
    M2 = multMatrix();
    render chair2;
    popMatrix();
    render chair1;
    popMatrix();
    render table;
    popMatrix();
    render room;
    popMatrix();
    render rug;
}
```
### Composite Transformations - Scaling

- **Observation 1:** The scale transformation also moves the object being scaled!

  Scale line segment from \( p_0 = (2,1,0,1)^T \) to \( p_1 = (4,1,0,1)^T \) to twice its length.

  ![Before and After Diagram](image1)

- **Observation 2:**

  Scaling segment from \( p_0 = (0,0,0,1)^T \) to \( p_1 = (2,0,0,1)^T \), left-hand endpoint does not move.

  ![Before and After Diagram](image2)

(\(0,0,0,1)^T\)) is known as a **fixed point** for the basic scaling transformation.

- Composite transformations to create desired transformation behavior.

### Composite Transformations

#### Fixed Point Scaling

We can use composite transformations to create a scale transformation with **different fixed points**, e.g., scale by 2 with fixed point = \((2,1,0,1)^T\)

1. Translate the point \((2,1,0,1)^T\) to the origin.
2. Scale by 2.
3. Translate origin to point \((2,1,0,1)^T\).

\[
\begin{bmatrix}
1 & 0 & 0 & 2 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 & -2 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix}
2 & 0 & 0 & -2 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
2 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix} C \end{bmatrix} = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

![Before and After Diagram](image3)
Composite Transformations
Fixed Point/Axis Rotation

Rotation of $\theta$ degrees about point $(x,y,z)^T$ in 2D generalizes to a rotation about a point $(x,y,z)^T$ and an axis $(x',y',z',0)^T$ in 3D:

1. Translate $(x,y,z,1)^T$ to origin
2. Rotate around axis $(x',y',z',0)^T$
3. Translate origin to $(x,y,z)^T$

Example for:

\[
\begin{align*}
(x,y,z,1)^T &= (1,3,0,1)^T \\
(x',y',z',0)^T &= (0,0,1,0)^T
\end{align*}
\]

\[
C = \begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 3 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & -1 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Composite Transformations
The Euler Transform

Assume, that the view looks
1. down the negative z-direction with
2. up in the positive y-direction and
3. right in the positive x-direction

\[
E(h, p, r) = R_z(r)R_x(p)R_y(h)
\]

$h=$head, $p=$pitch, $r=$roll

Example: $p=\pi/2$: \[
E(h, \pi/2, r) = R_z(r)R_x(\pi/2)R_y(h)
\]

\[
E(h, \pi/2, r) = \begin{bmatrix}
\cos r & -\sin r & 0 & 0 \\
\sin r & \cos r & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \frac{\pi}{2} & -\sin \frac{\pi}{2} & 0 \\
0 & \sin \frac{\pi}{2} & \cos \frac{\pi}{2} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos h & 0 & \sin h & 0 \\
0 & 1 & 0 & 0 \\
-\sin h & 0 & \cos h & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Composite Transformations
The Euler Transform

Example usage:
Limit a specific degree of rotational freedom to simulate physical behaviour, e.g., during mounting of screws.

This matrix is dependent on just one angle!

**Gimbal lock: one degree of freedom is lost!**

Example: $p=\pi/2$, then the $z$-rotation becomes rotation around $y$-axis.

\[
E(h, r) = R_z(r)R_y(\frac{\pi}{2})R_x(h)
\]
Composite Transformations

The Euler Transform

Extracting parameters $h,p,r$ from the orthogonal Euler transform (just using the upper 3x3):

$$F = \begin{bmatrix}
  f_{00} & f_{01} & f_{02} \\
  f_{10} & f_{11} & f_{12} \\
  f_{20} & f_{21} & f_{22}
\end{bmatrix} = R_z(r)R_y(p)R_x(h) = E(h,p,r)$$

where $\text{atan2}$ is the C-math function which avoids divide by 0.

- $h = \arctan\left(\frac{f_{21}}{f_{22}}\right)$
- $p = \arcsin\left(\frac{f_{01}}{f_{11}}\right)$
- $r = \arctan\left(\frac{-f_{01}}{-f_{02}}\right)$

$F$ with $p < -\frac{\pi}{2}$ or $p > \frac{\pi}{2}$

1. Original parameter can not be extracted!
2. $h,p,r$ are not unique!
Transformations
Rotation about an arbitrary axis

1. Assume rotation axis \( r \) which is normalized.
2. Rotate \( \alpha \) around \( r \).
3. Idea: Change bases, create base from \( r \).

\[ \bar{s} = \begin{cases} 
(0, -r_z, r_y), & \text{if } \left| r_x \right| < \left| r_y \right| \land \left| r_x \right| < \left| r_z \right| \\
(-r_x, 0, r_z), & \text{if } \left| r_y \right| < \left| r_x \right| \land \left| r_y \right| < \left| r_z \right| \\
(-r_x, r_y, 0), & \text{if } \left| r_z \right| < \left| r_x \right| \land \left| r_z \right| < \left| r_y \right| 
\end{cases} \]

\[ \frac{s}{s} = \bar{s} \]

\[ s = \frac{\bar{s}}{||\bar{s}||} \]

\[ t = r \times s \]

\[ M = \begin{bmatrix} r^T \\ s^T \\ t^T \end{bmatrix} \]

\[ X = M^T R_x(\alpha) M \]

Transformations
Rotation - quaternions

• Based on usage of complex numbers in 2D to express rotations:
  
  \[ z = x + yi = r(\cos \theta + \sin \theta) \]

• Let there be the pure imaginary number: \( i, i^2 = -1 \)
• Recall Euler’s identity: \( e^{i\theta} = \cos \theta + i \sin \theta \)
• then the polar representation of a complex number \( c \) is: \( c = a + ib = re^{i\theta} \)
• where \( r = \sqrt{a^2 + b^2}, \theta = \tan^{-1} \frac{b}{a} \)
• Rotating \( c \) about \( \phi \) degrees about the origin using the polar representation:
  
  \[ c' = re^{i(\theta + \phi)} = re^{i\theta} e^{i\phi} \]

\( e^{i\phi} \) is a rotation operator in the complex plane (and an alternative to transformations)
• But for 3D rotations we need a direction (vector) and degree (scalar).
• A suitable mathematical object is a quaternion:

\[ a = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q}), \text{ with } \mathbf{q} = (q_1, q_2, q_3) \]
Transformations
Rotation - quaternions

- Quaternion operations are based on three “complex” numbers with
  \[ i^2 = j^2 = k^2 = ijk = -1 \]
- which are analogous to unit vectors in 3D, hence for \( a = (q_0, q) \)
  \[ q = (q_0, i + q_2, j + q_3, k) \]
- Then for two quaternions \( a = (q_0, q), b = (p_0, p) \) addition is given as
  \[ a + b = (p_0 + q_0, p + q) \]
- and multiplication as \( ab = (p_0 q_0 - q \cdot p, q_0 p + p_0 q + p \times q) \)
- with the multiplicative identity being \((1,0)\)
- Quaternion magnitude is \( |a|^2 = q_0^2 + q_1^2 + q_2^2 + q_3^2 = q_0^2 + q \cdot q \)
- and the inverse as
  \[ a^{-1} = \frac{1}{|a|^2} (q_0, -q) \]

Transformations
Rotation - quaternions

- Relating the mathematical object to required geometric objects:
- Using the quaternion’s vector part as point in space: \( p = (0, p), p = (x, y, z) \)
- Consider (unit) quaternion: \( r = (\cos \frac{\theta}{2}, \sin \frac{\theta}{2} v) \)
- with \( v \) having unit length \(|v| = 1\) and hence: \( r^{-1} = (\cos \frac{\theta}{2}, -\sin \frac{\theta}{2} v) \)
- Then the product of \( p \) and \( r \) results in the quaternion \( p' = r pr^{-1} \) having the form \((0, p')\) which
  rotates the point \( p \) by \( \theta \) degrees around \( v \!
\[
p' = \cos^2 \frac{\theta}{2} p + \sin^2 \frac{\theta}{2} (p \cdot v) v + 2 \sin \frac{\theta}{2} \cos \frac{\theta}{2} (v \times p) - \sin \frac{\theta}{2} (v \times p) \times v
\]
Transformations
Rotation - quaternions

• Example: Rotating around the z axis with fixed point at origin and angle $\theta$:
  \[ v = (0,0,1) \Rightarrow r = \cos \frac{\theta}{2} + \sin \frac{\theta}{2} (0,0,1) \]

• with \[ p = (x, y, z) \Rightarrow p' = rp r^{-1} = r(0, p)r^{-1} = (0, p') \]

• where \[ p' = (x \cos \theta - y \sin \theta, x \sin \theta + y \cos \theta, z) \]
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 & 0 \\
  \sin \theta & \cos \theta & 0 & 0 \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
  \end{bmatrix}
  \]

• which is the same as
  

- Expected result with fewer operations!
- Avoids gimbal lock!
- Quaternions allow smooth interpolation between orientations.

Transformations
Normal transforms

• Object descriptions involve normals for lighting and shading calculation.

• What happens if we apply (composite) transformation $M$ to normals as we do it to vertices? Consider that $S_4(0.5)$ is a part of $M$:

1. Given a planar surface, a tangent vector is given by: \[ \vec{t}_1 = p_1 - p_0 \]
2. And hence a normal vector as: \[ \vec{n} = \vec{t}_1 \times \vec{t}_2 \]
3. Normal is orthogonal to tangent vectors by construction: \[ \vec{n} \cdot \vec{t} = \vec{n} T = 0 \]
4. Transforming a surface using a matrix: \[ p' = Mp \]
5. results in transforming the tangent vectors by the upper 3x3 matrix: \[ t' = M_{11}t \]

\[ \vec{n} \cdot \vec{t} = \vec{n} M_{11}^{-1} M_1 t^T = (M_{11}^{-1} \vec{n}^T) \cdot (M_1 \vec{t}^T) = \vec{n}^T \vec{t} = 0 \Rightarrow \vec{n}^T = \mathbf{N} \vec{n}^T = M_{11}^{-1} \vec{n}^T \]
Transformations

Normal transforms

- Normal vectors are transformed by the transpose of the inverse of tangents transformation!
- Well known from tensor algebra where
  - $\hat{\mathbf{t}}$ is called contravariant tensor of rank 1 and
  - $\hat{\mathbf{n}}$ is the covariant tensor of rank 1.

- Normal transforms require $\mathbf{N} = (\mathbf{M}^{-1})^T$ instead of $\mathbf{M}$
- Know the matrix, Neo!
- Try to avoid calculation of invert!
  - For an orthogonal matrix: $\mathbf{R} : (\mathbf{R}^{-1})^T = \mathbf{R}$
  - Translations do not affect normal directions.
  - Uniform scaling only alters normals’ lengths: can be renormalized if scaling is known.
  - Even in worse case only adjoint of upper 3x3 is needed (with later renormalization).