QUATERNIONS & ORIENTATION:
- Euler’s Theorem
- Quaternions & Vectors
- Orientation
- Coding

Prof. Christoph Hoffmann

Object Manipulation

- Translations are easy; simply add the displacement vectors.
- Rotations more complicated
  - How to build composite rotations?
  - After all, the local frame changes with each rotation
- For instance, what is intended by
  \begin{verbatim}
  glRotatef(30,0,1,0);
  glRotatef(30,0,0,1);
  ?
  \end{verbatim}
Example

First, rotate 30 degrees about $y$-axis,
Second, rotate 30 degrees about $z$-axis

But Suppose

What if rotation about $z$-axis should always mean rotate in the image plane?
Requires two steps:
1. Do the incremental (local) rotation
2. Apply it to the current orientation!

Problem: transforms are applied from last to first:
1. glLoadIdentity();
2. Transform*();
3. Transform*();
4. Transform*();
5. Draw object

Step 4 is applied first! So a simple sequence is not necessarily what we mean if we store angles…

And anyway, the sequence rotate about $z$-axis, and then the $y$-axis won’t pick the global frame for the second rotation
4x4 Transform

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>R</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>S=0</td>
<td>S=0</td>
<td>S=0</td>
<td>U=1</td>
<td></td>
</tr>
</tbody>
</table>

We need to work out the transforms!!!
Semantics

• Of the sequence:
  - `glRotatef(30, 0,1,0);`
  - `glRotatef(30, 0,0,1);`

• Means:
  - Construct the 4x4 transforms of the rotations
  - Multiply them to obtain the composite transform

Possible Strategy

• Store the current orientation of the object as a transform
• Apply that transform, then apply the incremental operation:
  • `glBegin(GL_MODELVIEW);`
  • `glLoadIdentity();`
  • `// multiply new transform here`
  • `// multiply stored orientation matrix`
  • `// render object`
But First…

- If we do only rotations in the \(xy\)-plane, in the manner described, can we orient the object in all possible ways?
- Yes:
  - Proof by construction
  - Note Euler angles

General Rotation

\[
R = \begin{bmatrix}
m_{11} & m_{12} & m_{13} & 0 \\
m_{21} & m_{22} & m_{23} & 0 \\
m_{31} & m_{32} & m_{33} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

- \(R\) is orthogonal:
  - \(R^{-1} = R^T\)
  - \(|R| = 1\)

- Euler’s Theorem, 1700’s –
  - Any combination of rotations about origin is equivalent to a \textit{single rotation} about some axis (through the origin)
  - Product of two rotation matrices is again a rotation matrix

Leonhard Euler
Quaternions
(1843)

Progression
• Real numbers, unit is 1
• Complex numbers, units are 1 and i
  (~1799, C. Wessel; C. F. Gauss)
• Quaternions, units are 1, i, j, and k

Multiplication rules:
• \( i \cdot i = j \cdot j = k \cdot k = -1 \)
• \( i \cdot j = -j \cdot i = k \)
• \( j \cdot k = -k \cdot j = i \)
• \( k \cdot i = -i \cdot k = j \)

Unit Quaternions

\[ a + bi + cj + dk, \text{ where } a^2 + b^2 + c^2 + d^2 = 1 \]

• Represent rotations about the origin
• Allow interesting motion interpolation
• Computationally no cheaper than 4x4 transforms
• Extension to rigid motions (Chasles’ theorem)
  – Uses dual quaternions (combined using Clifford algebra)
  – ME’s screw theory/calculus
Geometric Interpretation

\[
\cos(\theta/2) + ia\sin(\theta/2) + jb\sin(\theta/2) + kc\sin(\theta/2)
\]

\[
a^2 + b^2 + c^2 = 1
\]

\((a,b,c)\) is axis of rotation
\(\theta\) is the angle of rotation

Rotating Points with Quaternions

Points in 3D represented by coordinate vectors:
\[
p = bi + cj + dk \equiv (b, c, d)
\]
Think of it as a kind of quaternion…

Then simply multiply the unit quaternion \(R\) representing the rotation with the vector quaternion \(p\) so that
\[
p' = RpR^*
\]
where \(R^*\) is the conjugate of \(R\)
Then \(p'\) is the rotated point.
Example

Rotate \( p = (1,0,0) \) by \( 60^\circ \) about \( z \)-axis \((R)\):

\[ p = i, \ R = u + v k, \ R^* = u - v k \]

where \( u = \frac{1}{2}\sqrt{3}, \ v = 1/2, \)

so

\[ R p = u i + v k i = u i + v j \]
\[ R p R^* = u^2 i + u v j - u v k - v^2 j k \]
\[ = (u^2 - v^2) i + 2 u v j \]
\[ = v i + u j \]

\[ i^2 = j^2 = k^2 = -1 \]
\[ ij = -ji = k \]
\[ jk = -kj = i \]
\[ ki = -ik = j \]

Coding & OpenGL

- Since we have \( \text{glRotatelf}(\alpha, A_x, A_y, A_z) \) we can code directly how a unit quaternion acts:

\[ a + b i + c j + d k \]

becomes

\[ u = 2*\text{acos}(a); \]
\[ u = u*180/\pi; \]
\[ \text{glRotated}(u, b, c, d); \]

- We can also store the current and the new orientations as quaternions
Quaternion \( (q_0, q_1, q_2, q_3) \) to rotation matrix:

\[
\begin{pmatrix}
2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\
2(q_1q_2 - q_0q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2q_3 + q_0q_1) \\
2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2(q_0^2 + q_3^2) - 1
\end{pmatrix}
\]

3x3 Rotation matrix \( R_{jk} \) to quaternion:

\[
\begin{align*}
q_0^2 &= (R_{11} + R_{22} + R_{33} + 1) / 4 \\
q_1^2 &= (R_{11} - R_{22} - R_{33} + 1) / 4 \\
q_2^2 &= (R_{22} - R_{11} - R_{33} + 1) / 4 \\
q_3^2 &= (R_{33} - R_{11} - R_{22} + 1) / 4
\end{align*}
\]
Practicalities

• When composing rotations, numerical error can creep in and destroy the orthogonality of the matrices.
• Re-orthogonalizing a rotation matrix is more complicated than renormalizing a quaternion to unit length.
• Using quaternions instead of transforms does not reduce the number of arithmetic operations

Chasles’ Theorem

• Any two rigid body motions in 3 space can be replaced by a single screw motion.
• Analogous to Euler’s theorem
• Implies that a 4x4 rbt multiplied with another 4x4 rbt is again a 4x4 rigid-body transform.
• In a 4x4, what comes first, the translation or the rotation?