Introduction to Computer Graphics
Lecture Notes
3/4/2019

Rotations

1. Rotations are represented in 3x3 matrices. 9 numbers, but 3 are independent.

   a. Euler angles

      Pair of three angles relative to the axes

      Example:

      zyz – Refers to rotation angle about current frame.

      Rpy – Refers to rotation angle about a fixed frame

   b. Angle and Axis – You are given a general 3D vector V, and angle alpha

      We have seen that these representations have singularities, which means that the inverse problem cannot always be solved.

2. Inverse problem – Find the three independent rotation parameters that take one point to another point in space.

3. Unit Quaternions

   a. Stores four variables.

   b. The inverse problem can always be solved.

   c. The quaternion, Q = (Z,e), where Z is a variable and e is a vector.

      Z = cos(v/2), e = sin(v/2)r

      i. A quaternion is a rotation of Z degrees on the vector r.

      ii. v is the angle of rotation.

      iii. r is the axis of rotation.

$E = [e_x]$, $||r||^2 = 1 = r_x^2 + r_y^2 + r_z^2 = 1$

$[e_y]$ $||e||^2 = e_x^2 + e_y^2 + e_z^2 = \sin^2(v/2)(||r||^2) = \sin^2(v/2)$

$[e_z]$ $Z^2 = ||e||^2 = \cos^2(v/2) + \sin^2(v/2) = 1$

Q = (Z,e), $Z^2+e_x^2+e_y^2+e_z^2 = 1$. ← Unit quaternion

We have four variable and three independent variables. Given these four numbers, if we want the rotation matrix corresponding to the quaternion, we use this formula.

$$R(Z,e) = \begin{bmatrix} 2(Z^2+e_x^2)-1 & 2(e_x*e_y - Z*e_z) & 2(e_x*e_z + Z*e_y) \\ 2(e_x*e_y + Z*e_z) & 2(Z^2+ey^2)-1 & 2(e_y*e_z - Ze_x) \\ 2(e_x*e_z - Z*e_y) & 2(ey*ez + Z*e_x) & 2(Z^2+ez^2)-1 \end{bmatrix}$$

Lets look at the inverse problem using quaternions.

R = [r11 r12 r13], Z = ½(sqrt(r11+r22+r33+1))

[r21 r22 r23]

[r31 r32 r33]

Derivation:

r11 = 2(Z^2 +ex^2) -1}-> 2(Z^2 +ex^2 +Z^2 +ey^2 +Z^2 +ez^2) -3

r22 = 2(Z^2 +ey^2) -1}-> = 2(2Z^2 +1)-3

r33 = 2(Z^2 +ez^2) -1}-> = 4Z^2 +2-3 = 4Z^2-1

4Z^2 -1 = r11+r22+r33

Z^2 = ¼(r11+r22+r33+1)

Z = ½(sqrt(r11+r22+r33+1))

e = [sign(r32-r23)sqrt(r11-r22-r33+1)] [ex]

1/2[sign(r13-r31)sqrt(r22-r33-r11+1)] == [ey]

[sign(r21-r12)sqrt(r33-r11-r22+1)] [ez]

There are no divisions, so this representation has no singularities. Always use quaternion representation for rotations.

**sign(x) = 1 for x>=0, sign(x)=-1 for x<0

$Q = (Z,e) = R(Z,e)$.

$R^{-1}(Z,e) = Q^{-1}(Z,e) = (Z,-e)$

$RR^T = I \rightarrow R^{-1} = R^T$

4. Multiplying Rotation Matrices

$Q_1(Z_1,e_1) = R_1(Z_1,e_1)$

$Q_2(Z_2,e_2) = R_2(Z_2,e_2)$

$R_1*R_2 = R_1R_2$, which is expensive, because of 3x3 representation.

We want to calculate on four numbers to get the cumulative rotation.

Important identity: $Q_1*Q_2 = (Z_1*Z_2 - e_1^T*e_2, Z_1*e_1 + Z_2*e_1 + e_1xe_1)$

Suppose $Q_2 = Q_1^{-1}$

$Q_1*Q_2 = Q_1*Q_1^{-1} = I = (1,(0))$, (0) is a vector

$Q_1 = (Z,e)$, $Q_1^{-1} = (Z,-e)$

$Z_1*Z_2 - e_1^T*e_2 = Z^2+e^T*e = Z^2 + e_x^2+e_y^2+e_z^2 = 1$

$Z_1*e^2 + Z_2*e^1 + e_{1x}e_2 = -Ze + Ze - e_xe = 0$

5. Looking at the code for the transformation

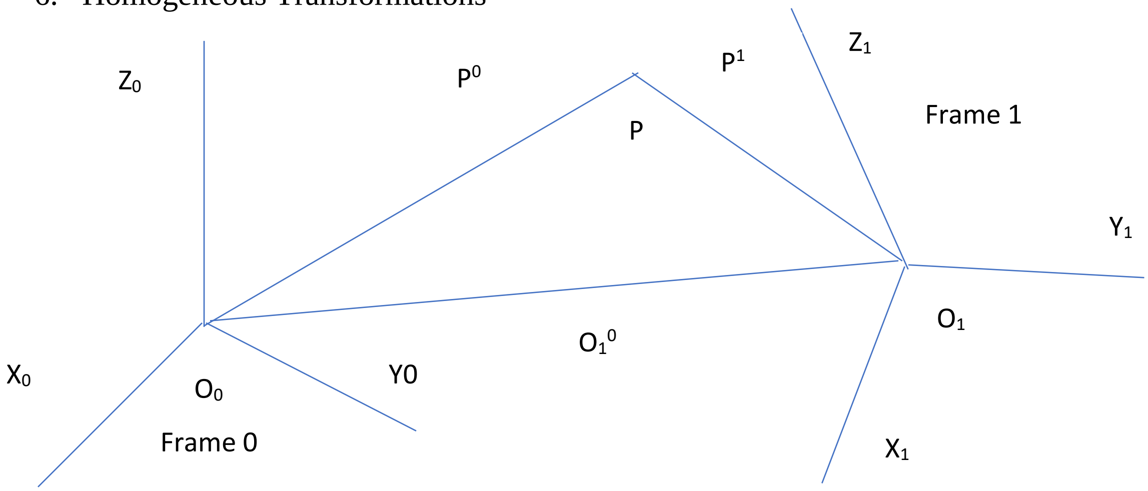
```
// generate transformation matrix
      glm::mat4 trans(1.0f);
      trans=glm::translate(trans,glm::vec3(0.5f,-0.5f,0.0f));

trans=glm::rotate(trans,glm::radians((GLfloat)glfwGetTime()*50.0f),glm::vec3(0.0f
,0.0f,1.0f));
```

Why do we use 4x4 matrices instead of a 3x3 matrix for rotation and a 3x1 vector for translation?

Introduction to Computer Graphics
Lecture Notes
3/4/2019

6. Homogeneous Transformations



i. Matrix-vector multiplication and vector-vector addition.

P^0 = $O_1^0 + R_1^0 * p^1$ this resorts to two different representations.

ii. Rather than this, suppose we create ~p which is $\begin{bmatrix} p_{3x1}^1 \\ 1 \end{bmatrix}$, $A_1^0 = \begin{bmatrix} R_{1,3x1}^0 & O_{1,3x1}^0 \\ 0^T & 1 \end{bmatrix}$

$*0^T = [0\ 0\ 0]$

The vertex shader has been using homogeneous transformations.

```
#version 330 core
Layout (location=0) in vec3 position;
layout (location=1) in vec3 color;

out vec3 our_color;

void main()
{
   gl_Position=vec4(position,1.0f);
   our_color=color;
}
```

$A_1^0 * \sim p = \begin{bmatrix} R_1^0 & O_1^0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} p^1 \\ 1 \end{bmatrix} = \begin{bmatrix} R_1^0 P^1 + O_1^0 \\ 1 \end{bmatrix}_{4x1} = \begin{bmatrix} p^0 \\ 1 \end{bmatrix}$ -> These are homogeneous coordinates

We want to perform matrix-vector multiplication ^^ instead of matrix-vector multiplication and vector-vector addition (P^0 = O1^0 + R1^0*p^1) to save space and time complexity.

iii. If we want to go from p^0 to p^1

$p^0 = O_1^0 + R_1^0 p^1$, $R_1^{0\wedge-1} = R_1^{0\wedge T}$

$R_1^{0\wedge-1} p^0 = R_1^{0\wedge T} p^0 = R_1^{0\wedge T} O_1^0 + (R_1^{0\wedge T} R_1^0 p^1) = R^{0\wedge T} O_1^0 + p^1$

- ➔ $p^1 = -R_1^{0\wedge T} O_1^0 + R_1^{0\wedge T} p^0$
- ➔ $A_0^1 = \begin{bmatrix} R_1^{0\wedge T} & -R_1^{0\wedge T} O_1^0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} p^0 \\ 1 \end{bmatrix} = \begin{bmatrix} p^1 \\ 1 \end{bmatrix}$

  $*\begin{bmatrix} p^n \\ 1 \end{bmatrix} = \sim p^n$
- ➔ $\sim p^1 = A_0^1 A_1^0 \sim p^1$
- ➔ $A_0^1 A_1^0 = I$
- ➔ $A_0^1 = A_1^{0\wedge-1}$

$A_0^1 \mathrel{!=} A_1^{0\wedge T}$, but $R_0^1 = R_1^{0\wedge T}$

Augmentation removes orthogonality of that matrix. Multiply by the inverse and not the transpose.