Quaternions

Comp 308

Rotation

At the moment we have rotations as matrices, or as angles around a vector. However, we are also being told that we need quaternions to represent rotations to do them properly. What is it that is hard about rotations?

You can consider naïvely rotating between two points on a sphere.

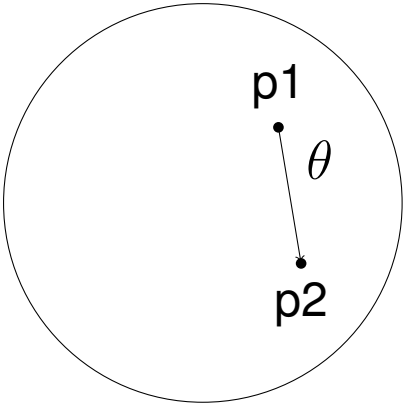

Figure 1: Linear rotation on a sphere

Rotation

To rotate sensibly on a sphere you need to move in a great arc. This is the equivalent of moving in a straight line if you are moving on a plane.

Quaternions are a way of representing rotations that ensures that all the rotations happen sensibly.

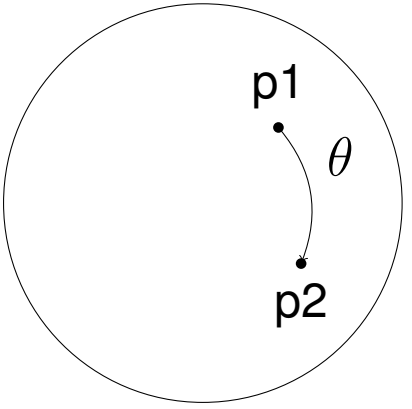

Figure 2: Great arc on a sphere

Quaternions

Quaternions are 4-tuples of real numbers ($\mathbb{R}$). Suppose we have a real number a. We can say a is a real number by writing $a \in \mathbb{R}$.

The set of all quaternions is written as $\mathbb{H}$. So we can say that q is a quaternion by writing $q \in \mathbb{H}$.

We also want to have vectors in 3D space. These are 3-tuples of real numbers $\mathbb{R}^3$. We write vectors in bold. We can write that $\mathbf{v}$ is a vector as $\mathbf{v} \in \mathbb{R}^3$

Notation

We can represent quaternions in a number of different ways. Despite this all the representations all mean the same thing. However, it is useful to have several different ways of writing them to make it easier.

We can represent them as a simple list of 4 elements:

$$q = [a, b, c, d] \qquad a, b, c, d \in \mathbb{R}$$

As the combination of a scalar, and a vector:

$$q = \langle a, \mathbf{v} \rangle \qquad a \in \mathbb{R}, \mathbf{v} \in \mathbb{R}^3$$

Or in their true form as an extension of complex numbers:

$$q = a + bi + cj + dk \qquad a, b, c, d \in \mathbb{R}$$

Assignment 3 Quaternions

For assignment 3 we have provided a quaternion class.

```
G308_Point p,q;
float angle = 90;
...
//Defined each of the parts in turn
quaternion r(1,0,0,0); // q(a, b, c, d)
//Defined as a rotation around p
quaternion r(angle, p);
//Defined as a rotation between two vectors
quaternion r(p, q);
```

Quaternion From Matrix

OpenGL treats matricies as a single array of length 16, in column order. So the matrix A would have the following indicies mapped to each element.

$$A = \begin{bmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{bmatrix}$$

So we can create a quaternion from an array representing a matrix as:

```
float matrix[16];
//Assign values to matrix
quaternion q(matrix);
```

Matrix From Quaternion

We can use our quaternions to get back matricies to use with OpenGL. Consider the following two use cases:

```
quaternion p;
...
float matrix[16];
p.toMatrix(matrix);

//replace the current matrix with p
glLoadMatrixf(matrix);

//Multiply the current matrix by p
glMultMatrixf(matrix);
```

Using the Quaternions

We have defined quaternions to be mostly **immutable** and they behave a lot like numbers. An example of what you can write in shown:

```
quaternion p,q,r,s;
//Assign the quaternions to things
quaternion pq = p * q;
pq = pq.multiplicativeInverse(); // (pq)^-1
pq = pq + r;
r = (0.5 * r)/s;
float d = dotproduct(p,q);
```

It is important to note that quaternions are immutable, and so you cannot actually change their value except by overwriting them. You also cannot directly access their internal representation, i.e., the 4 numbers that actually make them up. But you can get back the vector part with the `vector()` method.

Rotating a Vector Using Quaternions

Given a quaternion $q \in \mathbb{H}$ representing a desired rotation, and a vector $\mathbf{v} \in \mathbb{R}^3$ we can rotate $\mathbf{v}$ using q as follows

1. Make a new quaternion $v = \langle 0, \mathbf{v} \rangle$
2. Find $v' = \langle 0, \mathbf{v}'' \rangle = qvq^{-1}$

We then have that $\mathbf{v}''$ is $\mathbf{v}$ rotated by q.

Rotations Closed Under Rotation

Any ordered list of rotations can be represented by a single rotation.

Composing Rotations

So given two quaternions $p, q \in \mathbb{H}$, with each representing a rotation, how do we get the rotation of p then q?

Simple. Multiplication. pq is the rotation that is p followed by q.

This allows us to combine rotations by simply doing multiplication, which makes everything very easy.

Contents