

# Chapter 1

## Quaternions and Rotations

In this chapter we will focus on methods for manipulating rotations. In an earlier chapter, we discussed how a rotation can be described with a three by three matrix. In this chapter we will explore further how rotations can be represented and manipulated. In particular we shall look at how many numbers are needed to represent a rotation smoothly. We shall look at how rotations can be best interpolated. And we shall look at the arcball user interface for rotation. The key representation we shall explore in this context will be the quaternion representation.

### 1.1 Representations

We will begin by discussing various representations for rotations.

#### 1.1.1 matrices

As we have seen earlier, one can represent rotations using 3 by 3 matrices. Each rotation has a unique matrix representation. On the other hand, these matrices have 9 numbers which is more than we need.

Even though matrices have nine degrees of freedom, not all 3 by 3 matrices represent rotations. In particular a rotation is an orthonormal matrix with determinant  $+1$ . In an orthonormal matrix, each pair of two out of the three rows is orthogonal. This represents three constraints on the nine numbers. In an orthonormal matrix, the dot product of each of the rows with itself is 1. This represents three more constraints. So if we start with nine degrees of freedom, and restrict 6 of them, we should have 3 degrees of freedom remaining.

Of all the matrices satisfying these constraints, half of them have determinant  $-1$  and the other half have determinant  $+1$ . So the positive determinant constraint tosses

out half of the remaining matrices, but it does not reduce its dimension from three.

### 1.1.2 Euler Angles

From the above discussion, we may expect that there is a representation for rotations that uses only three numbers. Euler xyz angles are one such representation. Euler xyz angles represent an arbitrary rotation by specifying a sequence rotations about the x,y, and z axes. It can be shown that this can achieve any rotation.

One way to visualize the operation of Euler angles is using a set of gimbles. In such a device, there are three sets of axes as shown in Figure !!. Each axis is placed in some rotational setting. This will describe some rotation from the initial configuration.

Using the gimbles visualization, we can also see that for some set singular of rotations, Euler angles do not offer us a “unique” representation. For example, suppose one sets up the gimbles so that the inner  $z$  axis and outer  $x$  axis are lined up. Starting from this configuration, if one increases the rotation setting on the inner axis, one can exactly undo this by decreasing the rotation setting on the outer axis. Thus, there will be a whole family of Euler angels that will describe the exact rotation. This situation is called gimble lock, where there are three parameters but only two effective degrees of freedom.

### 1.1.3 Unit Quaternions

Quaternions give us a way of representing rotations using four numbers, avoiding any singular or special configurations. It is one more number than minimal, but there are no funny places.

A quaternion is simply a fourtuple of real numbers, and can be written as

$$\mathbf{Q} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

A quaternion can be written as a scalar  $w$  together with a 3 element coordinate vector  $\mathbf{v}$ .

$$\mathbf{Q} = \begin{bmatrix} w \\ \mathbf{v} \end{bmatrix}$$

A unit quaternions  $\hat{\mathbf{Q}}$  has unit norm where we define the norm as

$$\|\hat{\mathbf{Q}}\| = \sqrt{w^2 + x^2 + y^2 + z^2}$$

### Quaternions as representations

Here is how we will associate rotations to unit quaternions. Every rotation can be thought of as a right handed rotation of  $-2\pi < \theta < 2\pi$  radians about some axis. Let us represent the axis of rotation with a unit norm three element coordinate vector  $\hat{\mathbf{a}}$ . We will associate with this rotation the unit quaternion

$$\hat{\mathbf{Q}} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \hat{\mathbf{a}} \sin(\frac{\theta}{2}) \end{bmatrix}$$

For completeness, we mention here that the associated rotation matrix of a unit quaternion  $[w, x, y, z]^t$  is

$$\begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

Lets look at a few special examples. If  $\theta = 0$ , then this identity rotation is represented by the quaternion

$$\hat{\mathbf{Q}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

If I wish to represent the rotation of 180 degrees,  $\theta = \pi$  radians about  $\hat{\mathbf{a}}$ , the quaternion is

$$\hat{\mathbf{Q}} = \begin{bmatrix} 0 \\ \hat{\mathbf{a}} \end{bmatrix}$$

We call such rotations a “flip”.

### The Power Operator

We can define the power operator to represent scaling the rotation amount.

$$\begin{bmatrix} \cos(\frac{\theta}{2}) \\ \hat{\mathbf{a}} \sin(\frac{\theta}{2}) \end{bmatrix}^\alpha = \begin{bmatrix} \cos(\frac{\alpha\theta}{2}) \\ \hat{\mathbf{a}} \sin(\frac{\alpha\theta}{2}) \end{bmatrix}$$

As  $\alpha$  goes from from 0 to 1,  $\hat{\mathbf{Q}}^\alpha$  describes a series of rotations starting from the non-rotation ending at the complete rotation  $\hat{\mathbf{Q}}$ .

### Antipodal Quaternions

Suppose we have a rotation of  $\theta$  about  $\hat{\mathbf{a}}$

$$\hat{\mathbf{Q}} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \hat{\mathbf{a}} \sin(\frac{\theta}{2}) \end{bmatrix}$$

If we instead rotate about  $\theta - 2\pi$  we will get the exact same rotation. Writing down this quaternion we get

$$\begin{bmatrix} \cos(\frac{\theta-2\pi}{2}) \\ \hat{\mathbf{a}} \sin(\frac{\theta-2\pi}{2}) \end{bmatrix} = \begin{bmatrix} \cos(\frac{\theta}{2} - \pi) \\ \hat{\mathbf{a}} \sin(\frac{\theta}{2} - \pi) \end{bmatrix} = \begin{bmatrix} -\cos(\frac{\theta}{2}) \\ -\hat{\mathbf{a}} \sin(\frac{\theta}{2}) \end{bmatrix} = -\hat{\mathbf{Q}}$$

This is simply the negative of the original quaternion. So both  $\hat{\mathbf{Q}}$  and  $-\hat{\mathbf{Q}}$  represent the exact same rotation mapping. We call these two points antipodal.

The equivalence of antipodal unit quaternions as rotation mappings tells us that the topology of the space of rotation mappings is exactly the 3 dimensional projective space  $P^3$ . Recall, that  $P^3$  is obtained by taking all of the vectors in  $R^4$  and defining all scalar multiples (even negative ones) to be equivalent. A projective point in  $P^3$  contains exactly two antipodal vectors on  $S^3$ . Exactly like the space of rotation mappings.

### Powers of Antipodal Quaternions

Suppose I have two antipodal quaternions with  $\hat{\mathbf{Q}}$  written as a positive rotation,  $0 < \theta < 2\pi$ . If we apply the power operator to both  $\hat{\mathbf{Q}}$  and  $-\hat{\mathbf{Q}}$  we will get different behavior. For  $\alpha$  goes from 0 to 1,  $\hat{\mathbf{Q}}^\alpha$  gives us more and more rotation in the positive direction, while  $(-\hat{\mathbf{Q}})^\alpha$  gives more and more rotation in the negative direction. Both sequences start and end in the same place, but one of them will go the “long” way, with a flip in the center, while the other goes the “short” way.

If  $0 < \theta < \pi$  then then the positive rotation sequence is the short one. If  $\pi < \theta < 2\pi$ , then the negative rotation sequence is the short one. So the way to tell which quaternion will give the short short movie by looking at which one has a positive  $w$  coordinate.

### Application of Quaternions

Composition of rotations and applying a rotation to a vector can be accomplished directly in the quaternion representation.

If I give you two unit quaternions it turns out that I can obtain the quaternion representing the composition of their associate rotations by applying quaternion multiplication. Quaternion multiplication is defined as

$$\mathbf{Q}_1 \mathbf{Q}_2 = \begin{bmatrix} w_1 \\ \mathbf{v}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} (w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2) \\ (w_1 \mathbf{v}_2 + w_2 \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2) \end{bmatrix}$$

Under this definition of multiplication, we can verify that the inverse of a quaternion is obtained by negating its  $x, y, z$  values.

$$\begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}^{-1} = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix}$$

If I want to rotate some 3d vector represented by  $\mathbf{v}$  by some rotation represented by  $\hat{\mathbf{Q}}$ , it turns out that i can obtain the rotated vector represented by  $\mathbf{v}'$  by performing the following quaternion multiplication

$$\begin{bmatrix} 0 \\ \mathbf{v}' \end{bmatrix} = \hat{\mathbf{Q}} \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \hat{\mathbf{Q}}^{-1}$$

## 1.2 Interpolation of rotations

We saw earlier that in keyframe animation, the animator specifies the states of the scene objects at a sparse set of keyframes; each output frame is obtained by interpolating state values to intermediate values. We would like to understand how rotations can be interpolated just like positions and joint angles.

Suppose I have two rotations “a” and “b”, and I desire a rotation that is  $\alpha$  of the way from a to b. Let us call the output interpolated rotation “o”. The most obvious thing to try is to simply linearly interpolate the nine values of the matrixes representing the two rotations

$$M_o = (1 - \alpha)M_a + \alpha M_b$$

This turns out to be a horrible idea. The output matrix  $M_o$  is most likely not orthonormal, so it doesn’t even represent a rotation.

The next thing we may try is to simply linearly interpolate the three values representing the rotations in euler xyz angles

$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = (1 - \alpha) \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} + \alpha \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}$$

This is a bit better than the matrix entry approach since the resulting euler angles  $[x_o, y_o, z_o]^t$  must represent some rotation. The problem with this method that the resulting rotation is somewhat arbitrary. In particular if i first compose “a” and “b” by some other rotation “c”, and then interpolate, this method will not give me the composition of “c” and “o”.

We can solve this problem nicely using quaternions. Given the starting and ending quaternions  $\hat{\mathbf{Q}}_a$ , and  $\hat{\mathbf{Q}}_b$ . Let us define the “transition” rotation that takes from “a” to “b” as

$$\hat{\mathbf{Q}}_b \hat{\mathbf{Q}}_a^{-1}$$

We can now interpolate between “a” and “b” by starting with “a” and composing with it a scaled version of the transition using our power operator

$$\hat{\mathbf{Q}}_o = (\hat{\mathbf{Q}}_b \hat{\mathbf{Q}}_a^{-1})^\alpha \hat{\mathbf{Q}}_a$$

We have to be careful to make sure we are going the “short” way from a to b. To make sure of this, if our transition quaternion has a negative  $w$  value, then we must use the antipodal quaternion.

It can be shown that, unlike euler angle linear interpolation, this method of interpolation is invariant to compositions with a third rotation “c”. And so this is the preferred method of rotation interpolation.

This method is often called *slerp*, short for spherical interpolation. The reason for this is that the path of quaternions traced out by this method is a great arc on the unit sphere in  $R^4$ .

### 1.2.1 Splines of rotations

When we wanted to interpolate positions smoothly, we used spline based polynomial interpolation. We saw that these interpolations could be evaluated by doing repeated linear interpolation. This gives us an easy recipe for creating splines of rotations. One simply takes a spline method and replaces the linear interpolations with *slerp*s. It is just that easy.

In the previous chapter we discussed how one can use the catmull rom construction to obtain bezier control points  $\tilde{p}$  from a sequence of catmull rom control points  $\tilde{q}$ . Recall that we had

$$\begin{aligned} \tilde{p}_0 &= \tilde{q}_i \\ \tilde{p}_3 &= \tilde{q}_{i+1} \\ 3(\tilde{p}_1 - \tilde{p}_0) &= 1/2(\tilde{q}_{i+1} - \tilde{q}_{i-1}) \\ 3(\tilde{p}_3 - \tilde{p}_2) &= 1/2(\tilde{q}_{i+2} - \tilde{q}_i) \end{aligned}$$

We can apply the same construction to quaternions by analogy as

$$\begin{aligned} \hat{\mathbf{P}}_0 &= \hat{\mathbf{Q}}_i \\ \hat{\mathbf{P}}_3 &= \hat{\mathbf{Q}}_{i+1} \\ (\hat{\mathbf{P}}_1 \hat{\mathbf{P}}_0^{-1})^3 &= (\hat{\mathbf{Q}}_{i+1} \hat{\mathbf{Q}}_{i-1}^{-1})^{1/2} \\ (\hat{\mathbf{P}}_3 \hat{\mathbf{P}}_2^{-1})^3 &= (\hat{\mathbf{Q}}_{i+2} \hat{\mathbf{Q}}_i^{-1})^{1/2} \end{aligned}$$

There are many other ways of creating quaternion splines with better theoretical properties. These methods are beyond the scope of this book.

### 1.3 Arcball rotation interface

In an interactive computer graphics one often tracks the mouse and uses this as a way of specifying how to rotate and object. There are many ways to interpret mouse motion as rotations, and each of them have a slightly different feel for the user. In this section we will describe the arcball interface. The main advantage of this interface is that if the user starts moving the mouse at one point and finishes at another point, the final rotation will not depend on the path the mouse took in between.

Here is how the interface works.

- Imagine a hemisphere sitting on top of your window. You can decide where to center the hemisphere  $\tilde{c}$  and how big to make it. You can display the intersection of the hemisphere and the window by drawing the appropriate circle.
- When the user clicks on a pixel within the circle, that point is lifted up off the screen until it hits some point  $\tilde{p}_0$  on the imaginary hemisphere.
- Computer the vector  $\tilde{p}_0 - \tilde{c}$ , and normalize it to obtain  $\hat{v}_0$
- When the user now moves the mouse to some second point, repeat these steps to obtain  $\hat{v}_1$
- Imagine the arc along the hemisphere connecting  $\tilde{p}_0$  and  $\tilde{p}_1$ . The axis of rotation can be computed as  $\hat{a} = \hat{v}_0 \times \hat{v}_1$ . The angle  $\phi$  of the arc satisfies  $\cos(\phi) = \hat{v}_0 \cdot \hat{v}_1$
- Interpret this user interaction as a request to perform a rotation of  $\theta = 2\phi$  around  $\hat{a}$ .
- The unit quaternion that performs this rotation can be computed as

$$\begin{bmatrix} \cos(\frac{\theta}{2}) \\ \hat{a} \sin(\frac{\theta}{2}) \end{bmatrix} = \begin{bmatrix} \cos(\phi) \\ \hat{a} \sin(\phi) \end{bmatrix} = \begin{bmatrix} \hat{v}_0 \cdot \hat{v}_1 \\ \hat{v}_0 \times \hat{v}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{v}_1 \end{bmatrix} \begin{bmatrix} 0 \\ \hat{v}_0 \end{bmatrix}$$

Path independence of this interface can be shown easily. Suppose the user now moves the mouse to a new sphere point defining a new direction  $\hat{v}_2$ . What happens when we concatenate this next rotation request

$$\begin{bmatrix} \hat{v}_1 \cdot \hat{v}_2 \\ \hat{v}_1 \times \hat{v}_2 \end{bmatrix} \begin{bmatrix} \hat{v}_0 \cdot \hat{v}_1 \\ \hat{v}_0 \times \hat{v}_1 \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{v}_2 \end{bmatrix} \begin{bmatrix} 0 \\ \hat{v}_1 \end{bmatrix} \begin{bmatrix} 0 \\ \hat{v}_1 \end{bmatrix} \begin{bmatrix} 0 \\ \hat{v}_0 \end{bmatrix} = \begin{bmatrix} 0 \\ \hat{v}_2 \end{bmatrix} \begin{bmatrix} 0 \\ \hat{v}_0 \end{bmatrix}$$

Voila, we see that the intermediate request represented by  $\hat{v}_1$  become irrelevant!

The only wierd thing about this interface is that the imaginary arc of  $\phi$  radians is interpreted as a request for a rotation of  $\theta = 2\phi$ . For example if the user moves the

mouse from one end of the hemisphere across to the opposite end they are actually requesting a full 360 degree turn. This is the price we have to pay if we want path independence.

## 1.4 A rigid transformation class

In an earlier chapter, we discussed how 4 by 4 matrices could be used to represent the relationship between coordinate systems, such as eye, object, and world coordinate systems. Under user interaction, these matrices were updated by performing the appropriate matrix multiplication.

In this chapter we have discussed how quaternions are useful for representing rotations. They are useful to interpolate as well as to allow well behaved user interaction. Here we will discuss how to use quaternions in concert with translation vectors to represent rigid transformations.

A rigid transformation can be described by composing a translation and a rotation  $M = TR$ . This data can be encapsulated in a rigid transformation object made up of a single quaternion, and a single 3coordinate vector.

```
struct rigTform{
    coords3 t;
    quat r;
};

matrix4 gimeTRmatrix(rtf rigTform){
    matrix4 T = coords3ToTransMat(rigTform.t);
    matrix4 R = qrotToRotMat(rigTform.r);
    return matMatMul(T,R);
}
```

Given this representation, the relationship between an object's frame and the world frame is not maintained with a matrix  $O_i$ , but a `rigTform` object  $O_i$ .

remainder removed for class