Rotations and Interpolations

Technical Report

Adriana Schulz

Instructor: Luiz Velho

Rio de Janeiro, June 6, 2010
Contents

1 Overview 2

2 Technical Background 3
  2.1 Rotation Group ................................................. 3
  2.2 Rotation Representations ...................................... 5
    2.2.1 Euler angles ............................................. 5
    2.2.2 Rotation Matrices ....................................... 5
    2.2.3 Quaternions .............................................. 5
  2.3 Interpolating Rotations ...................................... 8
    2.3.1 SLERP .................................................. 8

3 Matlab Toolbox 10
  3.1 Description .................................................. 10
  3.2 Considerations ................................................ 11
    3.2.1 Magnitude ............................................... 11
    3.2.2 Data Representations .................................... 12

4 C++ Library 13
  4.1 Description .................................................. 13
    4.1.1 Euler Angles .......................................... 13
    4.1.2 Rotation Matrices ...................................... 14
    4.1.3 Quaternions ............................................. 14

References 15
Chapter 1

Overview

In this technical report we will discuss the different ways of representing rotations in the 3D-space. We will introduce the three most popular representations (Euler angles, rotation matrices and quaternions) and discuss rotation interpolation.

We will also describe the rotation package that was implemented both in Matlab and in C++.
Chapter 2

Technical Background

2.1 Rotation Group

The group which represents all possible rotations is $SO(3)$, which is the special orthogonal group in $\mathbb{R}^3$, i.e., the group of all orthogonal transforms which are positive. Notice that this set is isomorphic to the set of all $3 \times 3$ orthogonal matrices with positive determinants. $SO(3)$ is a group with multiplication, which guarantees that when two rotations are combined, the result is still a valuable rotation.

To prove that the set of $3 \times 3$ positive orthogonal matrices and the set of all possible rotations in $\mathbb{R}^3$ are equivalent, we will first consider a simplified version of Euler’s Theorem.

**Theorem 1** Let $A$ be a positive orthogonal operator in $\mathbb{R}^3$. Then there exists an orthogonal basis $E$ relative to which the matrix of $A$ can be written as:

$$
\begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) \\
0 & \sin(\theta) & \cos(\theta)
\end{bmatrix}
$$

For the demonstration, see Theorem 14.3 [1].

Theorem 1 guarantees that every element of $SO(3)$ represents a rotation of $\theta$ around the axis that is defined by the first element of the orthogonal basis $E$. On the other hand, rotations preserve length (isometry) and orientation. Every linear transformation that preserves inner products is represented by an orthogonal matrix (see Theorem 14.1 [1]). Moreover, a matrix
will preserve or reverse orientation according to whether the determinant of the matrix is positive or negative (proper or improper matrices). Therefore rotations are necessarily represented by positive orthogonal matrices.

We can justify the preservation of orientation by a continuity argument. If $S$ and $\tilde{S}$ are two coordinate systems, where $\tilde{S}$ can be obtained from $S$ by a continuous rigid motion of the coordinate axes, then the determinant of the matrix associated with the coordinate systems cannot change its value discontinuously.

It is important to point out that the fundamental system is arbitrary; it can be right–handed or left–handed. Nevertheless, consistency is fundamental. We have chosen to use the right-hand coordinate system (see Figure 2.1) in all functions implemented during this work (to be consistent with the BVH file description).

![Figure 2.1: The right-handed coordinate system.](image)

It is interesting to notice that $SO(3)$ is compact. In fact, it is limited because the columns of a matrix in $SO(3)$ have norm equal to one. Additionally, it is closed because it is the intersection of two closed sets: the sets of matrices with determinant equal to one and the set of orthogonal matrices (which are both closed because they are preimages of closed sets $\{1\}$ and $I$) by continuous functions.

Another important aspect to observe is that the space of rotations has three dimensions: two degrees of freedom to describe the rotation axis and one for the rotation angle.
2.2 Rotation Representations

There are several different ways to represent rotations. In this technical report we will study the three most important representations for computer graphics: Euler angles, rotation matrices and quaternions.

2.2.1 Euler angles

Euler angles take into account the fact that rotations have three degrees of freedom and describe them by three angles $\theta_x, \theta_y, \theta_z$, where each angle determines the rotation around each of the canonical axis.

Though this approach is quite efficient and largely used in computer graphics, it has a major drawback: gimbal lock. Gimbal lock is the loss of a degree of freedom with Euler angles, which occurs when two rotation axis align\(^1\). This phenomenon occurs because, as we discussed in the previous section, $SO(3)$ is compact and, therefore, any parametrization from an open set on $\mathbb{R}^3$ will necessarily involve singularities.

2.2.2 Rotation Matrices

As seen in section 2.1, rotations can be fully determined by a positive orthogonal matrix. Therefore it is natural to use rotation matrices to represent them. One of the drawbacks of this approach, however, is that a $3 \times 3$ matrices has nine degrees of freedom and therefore several constraints need to be satisfied in order to guarantee that a given matrix represents a rotation. In addition, when using rotation matrices, it necessary to store three times the necessary amount of data.

2.2.3 Quaternions

Quaternions are a very efficient way to represent rotations because they define an angle and a rotation axis.

It is not the intention of this work to fully demonstrate the properties of quaternions. Instead, we will introduce basic quaternion arithmetic and describe how they can be used to determine rotations.

\(^1\)We recomend [http://www.youtube.com/watch?v=zc8b2Jo7mno](http://www.youtube.com/watch?v=zc8b2Jo7mno) for an illustration of the phenomenon.
Quaternion Arithmetic

A quaternion $q \in \mathbb{R}^4$ can be defined as $q = w + xi + yj + zk$, where

\[
\begin{array}{c|ccc}
\cdot & i & j & k \\
\hline
i & -1 & k & -j \\
j & -k & -1 & i \\
k & j & -1 & -1
\end{array}
\]

Or equivalently, $q = q_0 + \mathbf{q}$, where $q_0$ is a scalar and $\mathbf{q}$ is a vector in $\mathbb{R}^3$.

Quaternion addition and multiplications are defined as follows:

\[
p + q = (w_p + w_q) + (x_p + x_q)i + (y_p + y_q)j + (z_p + z_q)k
\]

\[
p \cdot q = (p_0q_0 + \mathbf{p} \cdot \mathbf{q}) + (p_0q + q_0p + \mathbf{p} \times \mathbf{q})
\]

The set of quaternions, along with the operations of addition and multiplication, form a non-commutative division ring, where the multiplicative inversion is defined as follows:

\[
q^{-1} = \frac{q_0 - \mathbf{q}}{||q||}
\]

where,

\[
||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}
\]

Quaternion Rotation

Given a quaternion $q$, we represent a rotation as follows. If $\mathbf{v} \in \mathbb{R}^3$, let $v = 0 + \mathbf{v}$, then

\[
R_q(\mathbf{v}) = q\mathbf{v}q^{-1}
\]

Notice that $R_q(\mathbf{v})$ is also a quaternion with scalar value equal to zero,

\[
\Re(q\mathbf{v}q^{-1}) = \frac{(q\mathbf{v}^{-1})q\mathbf{v}q^{-1}}{2} = \frac{q(v + v^{-1})q^{-1}}{2} = q\Re(v)q^{-1} = 0
\]

where $\Re(q)$ is the real part of $q$. And, therefore, $R_q$ defines a function $\mathbb{R}^3 \to \mathbb{R}^3$.

2It is important to emphasize that the multiplication of quaternions is not commutative!
It is also simple to demonstrate that \( R_q \) is linear, orthogonal, and positive (see [2]), and, hence, we can conclude that \( R_q \) defines a rotation in \( \mathbb{R}^3 \).

Unitary quaternions are quaternions with norm equal to one. Notice that if \( q \) is not unitary, \( R_q = R_{\bar{q}} \), where \( \bar{q} = q/\|q\| \) (see Equation 2.1). Therefore, as far as rotations are concerned, we can always consider \( q \) unitary.

**Geometric Interpretation**

If \( q \) is unitary, then

\[
q_0^2 + |q|^2 = 1
\]

This implies that there exists \( \theta \), such that \(-\pi < \theta < \pi\) and

\[
\begin{cases}
q_0^2 = \cos^2 \theta \\
|q|^2 = \sin^2 \theta
\end{cases}
\]

And, therefore, if \( u = q/|q| \), then

\[
q = \cos \theta + u \sin \theta
\]

It is possible to prove that \( q \) defines a rotation of \( 2\theta \) around the \( u \) axis. We will not demonstrate this here (see [2]), however, we will illustrate the consistency of this result.

Notice that, since \( q \) is unitary, \( q^{-1} = \cos \theta - u \sin \theta \). Therefore \( q^{-1} = \cos(-\theta) + u \sin(-\theta) \), i.e., the inverse of the rotation of \( 2\theta \) is the rotation of \(-2\theta\).

We can also observe that, if \( u = 0 + u \):

\[
R_u(u) = (\cos \theta + u \sin \theta) u (\cos \theta - u \sin \theta)
= \cos^2 \theta u - \sin^2 \theta u^3
= (\cos^2 \theta + \sin^2 \theta) u = u
\]

i.e., rotating a point on the rotation axis does not affect it.\(^3\)

\(^3\)Notice that, if if \( q \) is unitary and \( q = q_0 + q \) is such that \( q_0 = 0 \), then

\[
q^2 = -q \cdot q + q \times q
= -|q|^2 = -1
\]
2.3 Interpolating Rotations

In several applications in computer graphics it is important to interpolate rotations. For example, a position of an articulate body is usually determined by the rotations of each joint and in keyframe animations are created by interpolating between these rotations.

Rotations represented by matrices are very difficult to be interpolated because simple linear combinations of the coefficients often result in matrices that are not positive orthogonal and therefore do not specify a valid rotation.

Euler Angles are more effective for this operation and are largely used for keyframe animation. The main drawback of this approach is that the interpolated frames will not necessarily be the most efficient path between keyframes since Euler angles are subject to gimbal lock problems (when two axis align). To get around gimbal problems, we can use different parametrizations (changing the order of the rotation axes). Nevertheless, changing between coordinate systems is difficult and computationally expensive.

The most efficient way of interpolating rotations is by means of quaternions. In order to understand quaternion interpolation, we will first introduce the relationship between quaternions and exponentials.

2.3.1 SLERP

Let \( q \) be a unitary quaternion, \( q = \cos \theta + u \sin \theta \). From the Taylor series we get

\[
e^{\theta u} = \cos \theta + u \sin \theta = q
\]

and, therefore, \( \log q = \theta u \).

Notice that,

\[
q^t = e^{t \log q} = e^{t \theta u} = \cos (t \theta) + u \sin (t \theta)
\]

Therefore, if \( c(t) = q^t \), where \( t \) varies between 0 and 1, then \( c(t) \) is a linear interpolation between \( p = 1 + 0i + 0j + 0k \) and \( q \).

Notice that the exponential is a parametrization of \( SO(3) : \mathbb{R}^3 \to S^3 \) and that the interpolation \( c(t) \) defines a geodesic on \( S^3 \). A geodesic is a desirable property because it guarantees that all inbetween values are valid rotations (on the great-circle) and the interpolation is linear (\( c'(t) \) is constant).

The Spherical Linear Interpolation (SLERP) is an extension of the previously described interpolation and is defined as follows:
\[
\text{slerp}(p, q, t) = p(p^{-1}q)^t
\]
where, \( p \) and \( q \) are quaternions and \( t \) is a parameter ranging from 0 to 1. Or, alternatively,

\[
\text{slerp}(p, q, t) = \frac{\sin((1 - t)\theta)}{\sin(\theta)}p + \frac{\sin(t\theta)}{\sin(\theta)}q
\]
where \( \theta \) is the angle between \( p \) and \( q \) \((\cos \theta = p \cdot q)\).

This interpolation is a geodesic that connects the mapping of the two points \( q \) and \( p \) on \( S^3 \).

Notice that SLERP only allows interpolating between two rotations. In order to consider more than two kewpoint it is necessary to consider different and more complex interpolations, however, will not address this issue in this work.
Chapter 3

Matlab Toolbox

We developed a couple of toolbox on Matlab that operates with quaternions and converts from quaternions to Euler angles and rotation matrices.

3.1 Description

The functions that implement operation with quaternions are:

- \( q_i = \text{Qinv}(q) \)
  returns a quaternion \( q_i \), which is the inverse of the quaternion \( q \).

- \( q = \text{Qmulti}(q_1, q_2) \)
  returns a quaternion \( q \), which is the result of the multiplication of quaternions \( q_1 \) and \( q_2 \).

- \( qt = \text{Qpow}(q, t) \)
  returns a quaternion \( qt \), which is the result of quaternions \( q \) raised to the power of \( t \).

- \( Pq = \text{Qrotation}(q, P) \)
  returns a 3D vector \( Pq \), which is the result of rotating the point \( P \) (also a 3D vector) by the rotation represented by the quaternions \( q \).

- \( q = \text{Qslerp}(q_1, q_2, t) \)
  returns a quaternion \( q \), which is the result of the Spherical Linear Interpolation (SLERP) of quaternions \( q_1 \) and \( q_2 \) with parameter \( t \), which varies from 0 to 1.
The functions that implement conversions between rotation representations are:

- \( q = \text{euler2quat}(e) \)
  returns a quaternion \( q \), which represents the same rotation given by the Euler angles \( e \).

- \( m = \text{euler2mat}(e) \)
  returns a rotation matrix \( m \), which represents the same rotation given by the Euler angles \( e \).

- \( q = \text{mat2quat}(m) \)
  returns a quaternion \( q \), which represents the same rotation given by the rotation matrix \( m \).

- \( e = \text{mat2euler}(m) \)
  returns the Euler angles \( e \), which represents the same rotation given by the rotation matrix \( m \).

- \( e = \text{quat2euler}(q) \)
  returns the Euler angles \( e \), which represents the same rotation given by the quaternion \( q \).

- \( m = \text{quat2mat}(q) \)
  returns a rotation matrix \( m \), which represents the same rotation given by the quaternion \( q \).

### 3.2 Considerations

#### 3.2.1 Magnitude

In computer graphics, we usually consider unitary quaternions (magnitude equals one). This simplifies calculations and has no negative effects since any scalar multiple of a quaternion results in the same rotation (see Equation 2.1). Therefore, in this toolbox, all functions were implemented considering unitary quaternions.
3.2.2 Data Representations

Rotation matrices are naturally represented as a $3 \times 3$ array. We represent Euler angles as a 3D vector, where the first element is the rotation around the $Z$ axis, the second element is the rotation around the $X$ axis, and the third element is the rotation around the $Y$ axis. A vector was also used to represent the quaternions $q = w + xi + yj + zk$. The first element represents $x$, the second $y$, the third $z$, and the forth $w$. 
Chapter 4

C++ Library

We also developed a Library in C++ that implements the functions presented in the previous section.

4.1 Description

The Library consists two header files (with corresponding sources): linearAlgebra.h and rotations.h. Examples of its use are given in the file main.cpp.

The file linearAlgebra.h contains two classes vector3f and matrix9f, which specify and implement the basic algebraic operations of a 3D vector and a $3 \times 3$ matrix, respectively\(^1\).

In the file rotation.h the three rotations representation classes are specified: Euler, RMatrix and Quaternion.

4.1.1 Euler Angles

Below is the description of the class Euler, which inherits from vector3f and describes rotations represented by Euler angles.

```cpp
class Euler : public vector3f
{
public:
    Euler();
```

\(^1\)The implementation was based on the corresponding classes from the free software Bioviewer.
The methods \texttt{euler2quat} and \texttt{euler2rmat} implement the conversion from Euler angles to quaternions and rotation matrices, respectively.

### 4.1.2 Rotation Matrices

Below is the description of the class \texttt{RMatrix}, which inherits from \texttt{matrix9f} and describes rotations represented by rotation matrices.

```cpp
class RMatrix: public matrix9f
{
  public:
    friend Euler rmat2euler (const RMatrix &m);
    friend Quaternion rmat2quat (const RMatrix &m);
}；
```

The methods \texttt{rmat2euler} and \texttt{rmat2quat} implement the conversion from rotation matrices to Euler angles and quaternions, respectively.

### 4.1.3 Quaternions

Below is the description of the class \texttt{Quaternion}, which describes rotations represented by quaternions.

```cpp
class Quaternion
{
  public:
    vector3f vector;
    float scalar;

    Quaternion();
    Quaternion(float x, float y, float z, float w);
```
Quaternion pow(float t);
Quaternion inv() const;
vector3f rot(vector3f P);

friend Quaternion operator+ (const Quaternion &q1, const Quaternion &q2);
friend Quaternion operator* (const Quaternion &q1, const Quaternion &q2);
friend Quaternion slerp (const Quaternion &q1, const Quaternion &q2, float t);

friend Euler quat2euler (const Quaternion &q);
friend RMatrix quat2rmat (const Quaternion &q);
};

Quaternions are represented by a scalar value (scalar) and a 3D vector (vector).

We implemented the basic algebraic operations of quaternions. The methods inv and pow implement quaternion inversion and the raise of a quaternion by a power of t, respectively. We also implemented operator overload to allow sum and multiplication of quaternions.

We also implemented rotations and interpolations of quaternions. The method rot returns a 3D vector, which is the result of rotating the point P by the rotation represented by the quaternions. The method slerp returns a quaternion q, which is the result of the Spherical Linear Interpolation (SLERP) of quaternions q1 and q2 with parameter t, which varies from 0 to 1.

Finally, the methods quat2euler and quat2rmat implement the conversion from quaternions to Euler angles and rotation matrices, respectively.
Bibliography


