CMSC 425: Lecture 7
3-Dimensional Rotations

Rotation and Orientation in 3-Space: One of the trickier problems 3-d geometry is that of representing rotations and the related problems of the orientation of frames. We have introduced the notion of orientation before (e.g., clockwise or counterclockwise). Here we mean the term in a somewhat different sense, as a directional position in space. Describing and managing rotations in 3-space is a somewhat more difficult task (at least conceptually), compared with the relative simplicity of rotations in the plane. We will explore two methods for dealing with rotation, *Euler angles* and *quaternions*.

Euler Angles: Leonard Euler was a famous mathematician who lived in the 18th century. He proved many important theorems in geometry, algebra, and number theory, and he is credited as the inventor of graph theory. Among his many theorems is one that states that the composition any number of rotations in three-space can be expressed as a single rotation in 3-space about an appropriately chosen vector. Euler also showed that any rotation in 3-space could be broken down into exactly three rotations, one about each of the coordinate axes.

Suppose that you are a pilot, such that the x-axis points to your left, the y-axis points ahead of you, and the z-axis points up (see Fig. 1). (This is the coordinate frame that most mathematical systems use.) Then a rotation about the x-axis, denoted by ϕ, is called the *pitch*. A rotation about the y-axis, denoted by θ, is called *roll*. A rotation about the z-axis, denoted by ψ, is called *yaw*. Euler's theorem states that any position in space can be expressed by composing three such rotations, for an appropriate choice of (ϕ, θ, ψ). Note that Unity swaps the z and y axes (see Fig. 1).

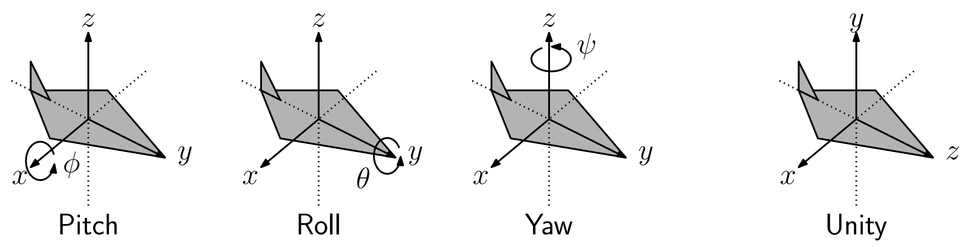

Fig. 1: Euler angles: pitch, roll, and yaw.

The order in which the rotations are performed is significant. In Unity (using the command transform.Rotate(x, y, z)), the order is the z-axis first (roll), x-axis second (pitch), and y-axis third (yaw). The x, y, and z values are given in degrees and the positive direction is clockwise. (Again, in traditional mathematics, positive rotations are performed in the counter-clockwise direction. This is probably a result of Unity's left-handed coordinate system.)

Shortcomings of Euler angles: While Euler angles are convenient for a single axis-aligned rotation, there are significant problems with their use for general rotations in 3-dimensional space. One issue is the fact that this representation depends on the choice of coordinate system. In the plane, a 30-degree rotation is the same, no matter what direction the axes are pointing (as long as they are orthonormal and right-handed). However, the result of an Euler-angle rotation depends very much on the choice of the coordinate frame and on the order in which the axes are named. (Later, we will see that quaternions do provide such an intrinsic system.)

Another problem with Euler angles is called *gimbal lock*. Whenever we rotate about one axis, it is possible that we could bring the other two axes into alignment with each other. (This happens, for example if we rotate x by $90°$.) This causes problems because the other two axes no longer rotate independently of each other, and we effectively lose one degree of freedom. Gimbal lock as induced by one ordering of the axes can be avoided by changing the order in which the rotations are performed. But, this is rather messy, and it would be nice to have a system that is free of this problem.

Quaternions: We will now delve into a subject, which at first may seem quite unrelated. But keep the above expression in mind, since it will reappear in most surprising way. This story begins in the early 19th century, when the great mathematician William Rowan Hamilton was searching for a generalization of the complex number system.

Imaginary numbers can be thought of as linear combinations of two basis elements, 1 and i, which satisfy the multiplication rules $1^2 = 1$, $i^2 = -1$ and $1 \cdot i = i \cdot 1 = i$. (The interpretation of $i = \sqrt{-1}$ arises from the second rule.) A complex number $a + bi$ can be thought of as a vector in 2-dimensional space (a, b). Two important concepts with complex numbers are the *modulus*, which is defined to be $\sqrt{a^2 + b^2}$, and the *conjugate*, which is defined to be $(a, -b)$. In vector terms, the modulus is just the length of the vector and the conjugate is just a vertical reflection about the x-axis. If a complex number is of modulus 1, then it can be expressed as $(\cos\theta, \sin\theta)$. Thus, there is a connection between complex numbers and 2-dimensional rotations. Also, observe that, given such a unit modulus complex number, its conjugate is $(\cos\theta, -\sin\theta) = (\cos(-\theta), \sin(-\theta))$. Thus, taking the conjugate is something like negating the associated angle.

Hamilton was wondering whether this idea could be extended to three dimensional space. You might reason that, to go from 2D to 3D, you need to replace the single imaginary quantity i with two imaginary quantities, say i and j. Unfortunately, this this idea does not work. After many failed attempts, Hamilton finally came up with the idea of, rather than using two imaginaries, instead using three imaginaries i, j, and k, which behave as follows:

$$i^2 = j^2 = k^2 = ijk = -1 \qquad ij = k,\ jk = i,\ ki = j.$$

Combining these, it follows that $ji = -k$, $kj = -i$ and $ik = -j$. The skew symmetry of multiplication (e.g., $ij = -ji$) was actually a major leap, since multiplication systems up to that time had been commutative.)

Hamilton defined a *quaternion* to be a generalized complex number of the form

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k.$$

Thus, a quaternion can be viewed as a 4-dimensional vector $\mathbf{q} = (q_0, q_1, q_2, q_3)$. The first quantity is a scalar, and the last three define a 3-dimensional vector, and so it is a bit more intuitive to express this as $\mathbf{q} = (s, u)$, where $s = q_0$ is a scalar and $u = (q_1, q_2, q_3)$ is a vector in 3-space. We can define the same concepts as we did with complex numbers:

Conjugate: $\mathbf{q}^* = (s, -u)$

Modulus: $|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = \sqrt{s^2 + (u \cdot u)}$

Unit Quaternion: $\mathbf{q}$ is said to be a unit quaternion if $|\mathbf{q}| = 1$

Quaternion Multiplication: Consider two quaternions $\mathbf{q} = (s, u)$ and $\mathbf{p} = (t, v)$:

$$\begin{aligned} \mathbf{q} &= (s, u) = s + u_x i + u_y j + u_z k \\ \mathbf{p} &= (t, v) = t + v_x i + v_y j + v_z k. \end{aligned}$$

If we multiply these two together, we'll get lots of cross-product terms, such as $(u_x i)(v_y j)$, but we can simplify these by using Hamilton's rules. That is, $(u_x i)(v_y j) = u_x v_y (ij) = u_x v_y k$. If we do this, simplify, and collect common terms, we get a very messy formula involving 16 different terms. (The derivation is left as an exercise.) The formula can be expressed somewhat succinctly in the following form:

$$\mathbf{qp} = (st - (u \cdot v), sv + tu + u \times v).$$

Note that the above expression is in the quaternion scalar-vector form. The first term $st-(u\cdot v)$ evaluates to a scalar (recalling that the dot product returns a scalar), and the second term $(sv+tu+u\times v)$ is a sum of three vectors, and so is a vector. It can be shown that quaternion multiplication is associative, but not commutative.

Quaternion Multiplication and 3-d Rotation: So far, everything we have said about quaternions seems to be purely abstract algebraic manipulations. We will show that in fact, they provide a natural way to encode 3-dimensional rotations. First, let us define a *pure quaternion* to be one with a 0 scalar component

$$\mathbf{p} = (0, v).$$

Any quaternion of nonzero magnitude has a multiplicative *inverse*, which is defined to be

$$\mathbf{q}^{-1} = \frac{1}{|\mathbf{q}|^2}\mathbf{q}^*.$$

(To see why this works, try multiplying $\mathbf{q}\mathbf{q}^{-1}$, and see what you get.) Observe that if $\mathbf{q}$ is a unit quaternion, then it follows that $\mathbf{q}^{-1} = \mathbf{q}^*$.

As you might have guessed, our objective will be to show that there is a relation between rotating vectors and multiplying quaternions. In order apply this insight, we need to first show how to represent rotations as quaternions and 3-dimensional vectors as quaternions. After a bit of experimentation, the following does the trick:

Vector: Given a vector $v = (v_x, v_y, v_z)$ to be rotated, we will represent it by the pure quaternion $(0, v)$.

Rotation: To represent a rotation by angle θ about a unit vector u, you might think, we'll use the scalar part to represent θ and the vector part to represent u. (This is called the *axis-angle representation* of a 3-dimensional rotation. Unfortunately, this doesn't quite work. After a bit of experimentation, you will discover that the right way to encode this rotation is with the quaternion $\mathbf{q} = (\cos(\theta/2), (\sin(\theta/2))u)$. (You might wonder, why we do we use $\theta/2$, rather than θ. The reason, as we shall see below, is that "this is what works.")

Rotation Operator: Given a vector v represented by the quaternion $\mathbf{p} = (0, v)$ and a rotation represented by a unit quaternion $\mathbf{q}$, we define the *rotation operator* to be:

$$R_{\mathbf{q}}(\mathbf{p}) \;=\; \mathbf{q}\mathbf{p}\mathbf{q}^{-1} \;=\; \mathbf{q}\mathbf{p}\mathbf{q}^{*}.$$

(The last equality results from the fact that $\mathbf{q}^{-1} = \mathbf{q}^*$, if $\mathbf{q}$ is a unit quaternion). We claim that the result of this operation will always be a pure quaternion, and so it is possible to interpret the result as a vector. In particular, this vector will be the result of applying the rotation $\mathbf{q}$ to v.

We will give a formal justification of this later, but for now, let's consider what this gives us. Let us apply the above quaternion multiplication rule and use the fact that $\mathbf{q}^{-1} = \mathbf{q}^*$ for a unit quaternion $\mathbf{q} = (s, u)$. Letting $\mathbf{p} = (0, v)$ denote the object to be rotated and expanding/simplifying we obtain:

$$R_{\mathbf{q}}(\mathbf{p}) \;=\; (0,\; (s^2 - (u \cdot u))v + 2u(u \cdot v) + 2s(u \times v)). \tag{1}$$

(We leave the derivation as an exercise, but a few nontrivial facts regarding dot products and cross products need to applied.) It is not obvious that this has anything to do with rotation, but later we will show that this corresponds exactly to rotating v about the axis u by θ degrees.

Quaternions in Unity: Unity supports an object called Quaternion that encapsulates a quaternion. You can generate a quaternion that performs a rotation by x degrees about a given 3-dimensional vector $\vec{u}$ using the command Quaternion.AngleAxis(x, u). For example, the following command sets the current object's rotation to a rotation by a 30° about the vertical axis.

```
transform.rotation = Quaternion.AngleAxis(30, Vector.up);
```

There are a number of useful operations defined on quaternions, such as converting Euler angles to quaternions, multiplying quaternions, interpolating between quaternions, and computing a frame that is oriented with a quaternion. A common operation is generating a rotation that will cause an game object (e.g., a camera or weapon) to point towards a certain direction.

```
Vector3 relativePos = target.position - transform.position;
Quaternion rotation = Quaternion.LookRotation(relativePos);
transform.rotation = rotation;
```

Example: Consider the 3-d "roll" rotation shown in Fig. 2. This rotation can be achieved by performing a rotation about the y-axis by $\theta = 90$ degrees. Thus $\theta = \pi/2$, and the axis of rotation is $\widehat{u} = (0, 1, 0)$, and so we have $s = \cos(\theta/2) = 1/\sqrt{2}$ and $u = (\sin(\theta/2))\widehat{u} = (0, 1/\sqrt{2}, 0)$, and hence

$$\mathbf{q} \;=\; (\cos(\theta/2), (\sin(\theta/2))u) \;=\; \left(\cos\left(\frac{\pi}{4}\right), \sin\left(\frac{\pi}{4}\right)(0,1,0)\right) \;=\; \left(\frac{1}{\sqrt{2}}, \left(0, \frac{1}{\sqrt{2}}, 0\right)\right).$$

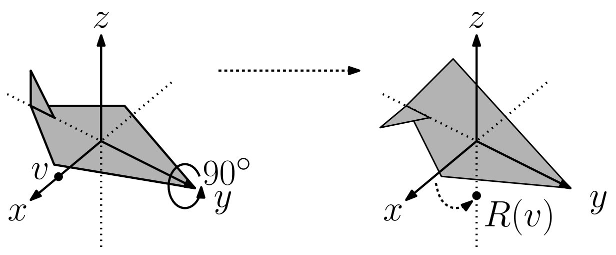

Fig. 2: Rotation example.

Let us consider how the x-unit vector $v = (1, 0, 0)$ is transformed under this rotation. To reduce this to a quaternion operation, we encode v as a pure quaternion $\mathbf{p} = (0, v) = (0, (1, 0, 0))$. Observe that

$$s^2 - (u \cdot u) = \frac{1}{2} - \frac{1}{2} = 0, \qquad (u \cdot v) = 0, \qquad \text{and} \qquad (u \times v) = \left(0, 0, \frac{-1}{\sqrt{2}}\right).$$

By applying the rotation operator, by Eq. (1), we have

$$\begin{aligned} R_{\mathbf{q}}(\mathbf{p}) &= (0,\ (s^2 - (u \cdot u))v + 2u(u \cdot v) + 2s(u \times v)) \\ &= (0,\ 0v + 2u0 + 2s(0, 0, -1/\sqrt{2})) \\ &= (0,\ \vec{0} + \vec{0} + (2/\sqrt{2})(0, 0, -1/\sqrt{2})) \\ &= (0, (0, 0, -1)). \end{aligned}$$

Interpreting $\mathbf{p}$ as a vector $(0, 0, -1)$, we see that, as expected, quaternion rotation rotates the vector $v = (1, 0, 0)$ by 90° to $(0, 0, -1)$ (see Fig. 2).

Why Quaternions Work: (Optional) In order to understand why the above quaternion operation implements rotation, we begin with the concept of *angular displacement*, which involves rotating a given vector v about a given rotation axis u (any unit vector) by a certain number of degrees θ.

Let $R(v)$ denote this rotated vector (see Fig. 3(a)). In order to derive this, we begin by decomposing v as the sum of its components that are parallel to and orthogonal to u, respectively.

$$v_{\|} = (u \cdot v)u \qquad v_{\perp} = v - v_{\|} = v - (u \cdot v)u.$$

Note that $v_{\|}$ is unaffected by the rotation, but $v_{\perp}$ is rotated to a new position $R(v_{\perp})$. To determine this rotated position, we will first construct a vector that is orthogonal to $v_{\perp}$ lying in the plane of rotation.

$$w = u \times v_{\perp} = u \times (v - v_{\|}) = (u \times v) - (u \times v_{\|}) = u \times v.$$

The last step follows from the fact that u and $v_{\|}$ are parallel, and so the cross product is zero. Clearly w is orthogonal to both $v_{\perp}$ and u. Furthermore, because $v_{\perp}$ is orthogonal to the unit vector u, it follows from basic properties of the cross product that w is the same length as $v_{\perp}$.

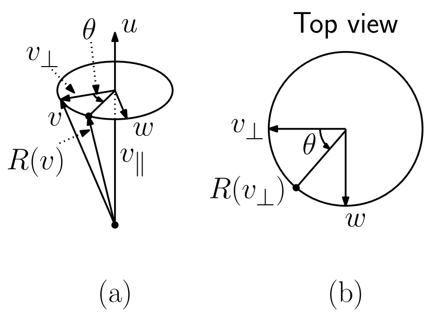

Fig. 3: Angular displacement.

Now, consider the plane spanned by $v_\perp$ and w (see Fig. 3(b)). We have

$$R(v_\perp) \;=\; (\cos\theta)v_\perp + (\sin\theta)w.$$

From this and the fact that $R(v_\|) = v_\|$, we have

$$\begin{aligned} R(v) &= R(v_\|) + R(v_\perp) \;=\; v_\| + (\cos\theta)v_\perp + (\sin\theta)w \\ &= (u\cdot v)u + (\cos\theta)(v - (u\cdot v)u) + (\sin\theta)w \\ &= (\cos\theta)v + (1-\cos\theta)u(u\cdot v) + (\sin\theta)(u\times v). \end{aligned}$$

In summary, we have the following formula expressing the effect of the rotation of vector v by angle θ about a rotation axis u:

$$R(v) \;=\; (\cos\theta)v + (1-\cos\theta)u(u\cdot v) + (\sin\theta)(u\times v). \qquad (2)$$

This expression is the image of v under the rotation. Notice that, unlike Euler angles, this is expressed entirely in terms of intrinsic geometric functions (such as dot and cross product), which do not depend on the choice of coordinate frame. This is a major advantage of this approach over Euler angles.

Now that we know how to express rotation in terms of vector operations, let's see how this relates to the quaternion rotation operation. Let us see if we can express this in a more suggestive form. Since $\mathbf{q}$ is of unit magnitude, we can express it as

$$\mathbf{q} = \left(\cos\frac{\theta}{2}, \left(\sin\frac{\theta}{2}\right)u\right), \qquad \text{where } \|u\| = 1.$$

Plugging this into Eq. (1) and applying some standard trigonometric identities, we obtain

$$\begin{aligned} R_{\mathbf{q}}(\mathbf{p}) &= \left(0, \left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right)v + 2\left(\sin^2\frac{\theta}{2}\right)u(u\cdot v) + 2\cos\frac{\theta}{2}\sin\frac{\theta}{2}(u\times v)\right) \\ &= (0,\ (\cos\theta)v + (1-\cos\theta)u(u\cdot v) + \sin\theta(u\times v)). \end{aligned}$$

Observe that the vector part of this quaternion is *identical* to the angular displacement equation for $R(v)$ presented in Eq. (2), implying that the quaternion rotation operator achieves the desired rotation.

Composing Rotations: (Optional) We have shown that each unit quaternion corresponds to a rotation in 3-space. This is an elegant representation, but can we manipulate rotations through quaternion operations? The answer is yes. In particular, the action of multiplying two unit quaternions results in another unit quaternion. Furthermore, the resulting product quaternion corresponds to the composition of the two rotations. In particular, given two unit quaternions $\mathbf{q}$ and $\mathbf{q}'$, a rotation by $\mathbf{q}$ followed by a rotation by $\mathbf{q}'$ is equivalent to a single rotation by the product $\mathbf{q}'' = \mathbf{q}'\mathbf{q}$. That is,

$$R_{\mathbf{q}'} R_{\mathbf{q}} = R_{\mathbf{q}''} \qquad \text{where } \mathbf{q}'' = \mathbf{q}'\mathbf{q}.$$

This follows from the associativity of quaternion multiplication, and the fact that $(\mathbf{q}\mathbf{q}')^{-1} = \mathbf{q}^{-1}\mathbf{q}'^{-1}$, as shown below.

$$\begin{aligned} R_{\mathbf{q}'}(R_{\mathbf{q}}(\mathbf{p})) &= \mathbf{q}'(\mathbf{q}\mathbf{p}\mathbf{q}^{-1})\mathbf{q}'^{-1} \ = \ (\mathbf{q}'\mathbf{q})\mathbf{p}(\mathbf{q}^{-1}\mathbf{q}'^{-1}) \\ &= (\mathbf{q}'\mathbf{q})\mathbf{p}(\mathbf{q}\mathbf{q}')^{-1} \ = \ \mathbf{q}''\mathbf{p}\mathbf{q}''^{-1} \\ &= R_{\mathbf{q}''}(\mathbf{p}). \end{aligned}$$