Issue 31 | Dec 2010

Blender learning made easy

blenderart MAGAZINE

Under the Microscope

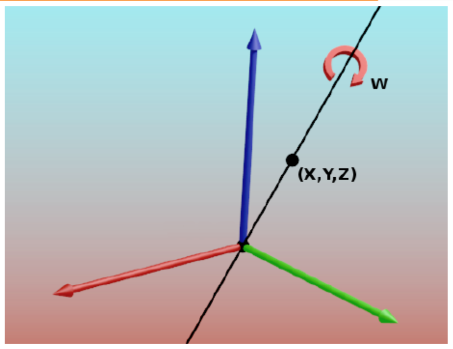
Physics of Circular Motion

A World of Rotations

BioBlender: Blender for Biologists

Computer Simulation and Modeling of Liquid Droplets

COVERART Virus -by Adam Auksel



by- Pep Ribal

Introduction

What is it with rotations that makes them so frightening?

Actually rotations are very useful, and sometimes absolutely necessary. Imagine a world without rotations... Tire manufacturers will agree with me... Indeed, of the three kinds of transformations (translation, rotation and scaling), rotations are by far the most complex. Let's see why.

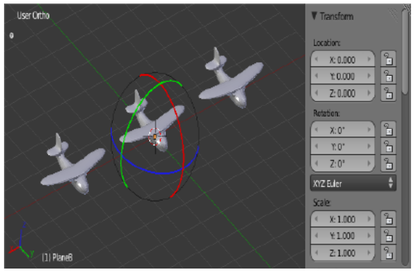
Take the Blender default scene. Activate the Transform panel (hotkey **N**). Make sure the Translate manipulator is on, and Transform Orientation is set to Global. Now move the default cube along the X axis using the manipulator (red arrow). Take a look at the Transform panel as you drag the cube. You'll see the Location X value change as you do, while the other values remain unchanged. OK, drop the cube where you wish. Now do the same along the Y axis, and you'll see Location Y change as you drag. Once more, the rest of the values remain unchanged. Finally, you can see the same thing happens with Z axis.

Then you can change the 3D manipulator to Scale. Keep trying with the three axes, and you'll realize that each modification affects only its own axis value (Scale values). It also changes the Dimensions values, but these are not relevant, as they refer to the final dimensions of the mesh, not to the transform properties of the object.

Rotating an object

First of all, a brief description of the Transform Orientations available for the 3D manipulators in Blender. View has a set of axes aligned with the viewport direction, Normal is aligned with the normal of the actual object data selection (like mesh faces) in Edit Mode, and it's equivalent to Local orientation in Object Mode, Local is aligned with the object local coordinate system, and Global is aligned to the world coordinate system. We will see later what Gimbal means.

Once that's said, let's start the show. First make sure XYZ Euler is selected in the Transform panel. Try now with the Rotate manipulator, with Global orientation. Drag around the Z axis (blue ring). You can also use hotkey R, and then Z for rotation around the Z global axis. You can see the Rotation Z value change as you rotate. Drop it at will. Now rotate around any of the other two axes... What happens? All three rotation values (X, Y and Z) change as you drag...



Location:
X: 0.000
Y: 0.000
Z: 0.000

Rotation:
X: 0°
Y: 0°
Z: 0°

XYZ Euler

Scale:
X: 1.000
Y: 1.000
Z: 1.000

Figure 1. Initial state

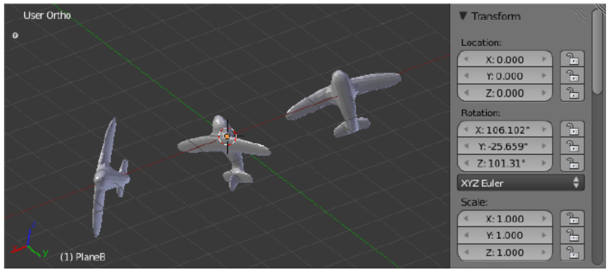
We have just discovered that rotation around one axis affects the value of the other two. Let's go deeper into this. Open the provided file 'RotationsWorld.blend'. There you have three simple airplanes (fig 1).

We will use the Rotate manipulator to perform three rotations on them: 120º around the X global axis, 60º around Y, and 45º around Z. But we will change the order of those rotations in each object.

Bear in mind that positive angles mean counterclockwise rotations.

Start with 'PlaneA'. Use the manipulator to rotate X axis first. Check the amount of rotation in the 3D view header, not in the Transform panel. Use the **Ctrl** key to round up the rotated value to 120º as you drag. If you use hotkeys **R** and then **X**, you can also enter 120 with the keyboard. Next, rotate 60º around Y axis and finally 45º around Z axis.

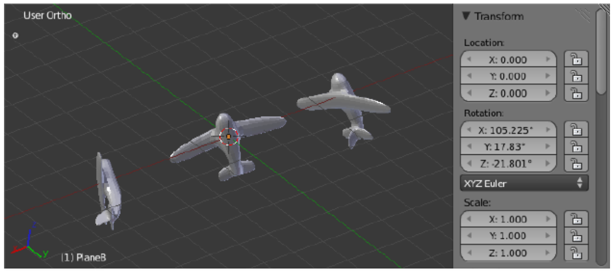
Now perform the same rotations on 'PlaneB' but in this order: first 60º around Y, then 120º around X, and last, 45º around Z. When done, go for 'Plane C', using a new order: 45º around Z, 120º around X, and 60º around Y (figure 2). Remember to always check the rotation amount in the 3D view header only.



Location:
X: 0.000
Y: 0.000
Z: 0.000
Rotation:
X: 106.102°
Y: -25.659°
Z: 101.31°
XYZ Euler
Scale:
X: 1.000
Y: 1.000
Z: 1.000

Figure 2. After rotations around global axes.

OK, what do we have now? Three airplanes with a completely different orientation in space. If you take a look at the rotation values of the three planes, only 'PlaneA' keeps the values of the applied rotation (X=120º, Y=60º, Z=45º), while the others hold very strange numbers. You can see that the order of rotation is important. Even if we use Local mode for rotation manipulators, the problem doesn't improve (figure 3). For rotation



Location:
X: 0.000
Y: 0.000
Z: 0.000
Rotation:
X: 105.225°
Y: 17.83°
Z: -21.801°
XYZ Euler
Scale:
X: 1.000
Y: 1.000
Z: 1.000

Figure 3. After rotations around local axes.

around X local axis for instance, you can also press **R, X, X.**

In translation and scaling we can just enter the values we wish into the Transform panel manually, as there is only one way to interpret their meanings. But as we have just seen, with rotations, entering the values X=120º, Y=60º, Z=45º in the slider controls might not yield the desired result. If we were looking for the orientation of 'PlaneA', that would have been OK. But if we wanted for instance, the rotation of any of the other two, that wouldn't have done the trick.

We need a rotation system with a special set of axes that lets us forget about the order, so that we can type the three rotation angles directly in the Transform panel, or use a manipulator so that each ring affects only one axis value. And that's exactly what Blender does. It's not using the global or local axes, as you have seen by the strange numbers you got in the rotation values of the objects. So what is that wonderful system that Blender uses internally?

Euler rotations

We have previously set rotation mode to XYZ Euler. That is exactly what Blender is using internally. The best way to see these type of rotations in action is to set the Transform Orientation of the 3D manipulator to Gimbal. This widget lets you see the current state of the Euler rotation transform.

A gimbal is a circular gadget that spins around an axis that goes through one of its diameters. If you mount three of these one inside the other, you have a 3-axes gimbal (figure 4). This kind of device is used in gyroscopes, for instance.

The Blender Gimbal rotation manipulator closely resembles one of these gadgets, as in a 3-axes gimbal these move in relation to the others. However, Blender gimbal is a bit different from that, the main difference being the axis of rotation of the rings as you can see in figures 5 and 6. While the physical gimbal rotates around one of its diameters (figure 5), each ring of the Blender gimbal rotates around an axis that goes through the centre of the ring and is perpendicular to all of its diameters (figure 6).

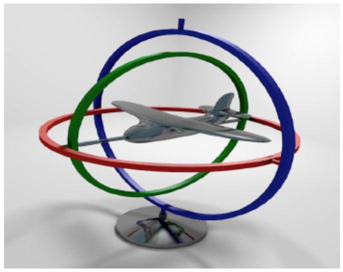


Figure 4. A gimbal gadget.

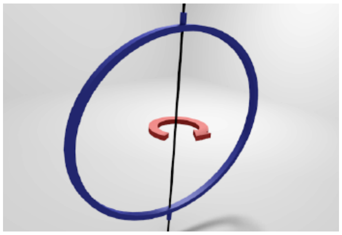


Figure 5. Physical gimbal rotation.

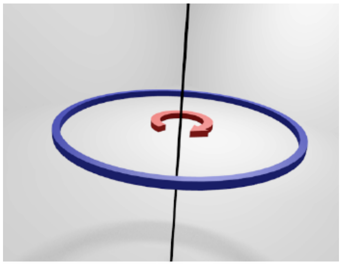
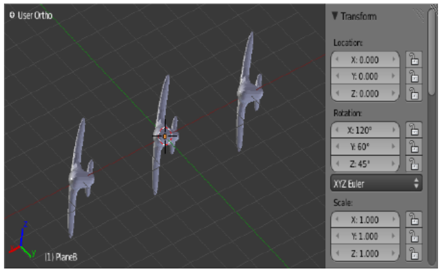


Figure 6. Blender gimbal rotation.



Transform
Location:
X: 0.000
Y: 0.000
Z: 0.000
Rotation:
X: 120°
Y: 60°
Z: 45°
XYZ Euler
Scale:
X: 1.000
Y: 1.000
Z: 1.000

Figure 7. After Euler rotations.

So, let's start playing with the gimbal. Take any object with 0 rotation. Now activate the Gimbal manipulator. Set the rotation mode to XYZ Euler (though it would work with any other Euler type). And now start rotating the axes individually. You can repeat the experiment of the three airplanes, and you'll get the results of figure 7. See what happens in the Transform panel.

Now, each gimbal axis is directly related to the corresponding rotation value of the object. What does it mean? That order of rotation doesn't matter. Maybe you have realized that all three airplanes end up in the same position using the Euler gimbal. If so, you will have seen that all three airplanes have the same rotation values in the Transform panel. In other words, you can enter the desired rotation angles numerically in the slider controls.

So, what's the difference between a local or global rotation system and the gimbal system? And why are there six different types of Eulers? And why am I asking all this if I know the answer...?

As you can see in a physical 3-axis gimbal gadget, there are three axes configured in such a way that they form a hierarchy. When we rotate the outermost ring, the one on top of the hierarchy, we are actually rotating the entire system around the axis of that ring.

Rotating the middle ring, we can see the innermost ring rotate as well. If we rotate the innermost ring, only that ring moves.

With Blender gimbals the same thing happens. So we must choose one axis to be on top of the hierarchy, one to be in the middle, and the last one to be at the bottom.

Let's say we want the Z axis to be on top; the X axis to be its child; and finally Y to be at the bottom. In other words, Z axis will be the parent of X, and X the parent of Y. In order from bottom to top, we have Y, X, and Z. That forms a YXZ gimbal, used for YXZ Euler rotations.

There are six different combinations of hierarchies with the three axis X, Y and Z, and therefore, six different kinds of gimbals, each one of which is associated to its corresponding Euler rotation system. With Eulers, it's important to remember that the axis written first is the one at the bottom of the hierarchy, while the last one on the right is the one on top of it. Thus, in a XYZ Euler, the Z axis in on top, while X is at the bottom.

Blender uses two things to calculate the Euler rotation of an object. First, the values of the three rotations around each of the three axes (X,Y and Z); and second, what type of Euler hierarchy are these values based on. For instance it's not the same to use a XYZ or a ZXY hierarchy. You can check this by taking the rotated airplane. Don't change the rotation values, just change to any of the other five Euler modes. You will immediately see the final rotation changes.

When Blender has calculated the rotation of the object (using the Eulers), it stores that rotation in the object matrix, which is basically a 4x4 matrix of numbers that keep track of the full transformation state of the object: its location, rotation and size. When you are just modelling (not animating), it doesn't make any difference which rotation mode you are using, as they will all end up in the same place internally, i.e. the matrix. No use will be made of the Euler values. However when animating, Blender actually uses those Euler values to interpolate rotations, as we will see later.

Any of the Euler types has the advantages of isolating the effect of each axis, though they yield different rotations. Not a big deal. It's just a question of experimenting with them, and see how each type of gimbal behaves. OK, now we have found a magical rotation system that will make this world a better place for you and me... So, why do we need other rotation systems?

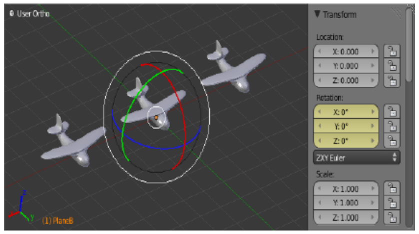
Euler rotation problems

If we want to define any orientation, or we want to rotate a face or a group of vertices, we can get them to rotate wherever we want to use any of the Euler modes. But when it comes to animation, we can run into some trouble in certain circumstances.When you want to animate a rotating object you have to use the same system from one keyframe to the other. You cannot start defining a XYZ Euler orientation for one keyframe, and then a YZX Euler for the next one. Why?

Because Blender interpolates between two rotations using the values of the specific system used (Euler or any other); it doesn't use the rotation stored in the object matrix. So if you use a different system in two consecutive keyframes, there is no way to calculate the interpolation values between them.

Let's go for another experiment. Open the file 'RotationsWorld.blend'. Make sure that the airplanes don't have any rotation applied, then go to frame 1 and select ZXY Euler rotation mode. We will focus on one of the airplanes as we are going to make it do some aerobatics. You can delete the other two if you want.

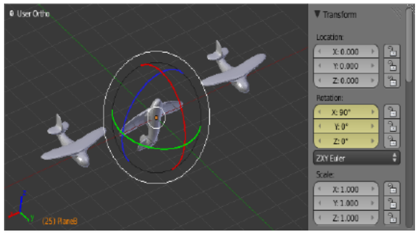
The first thing to do will be to set the Transform Orientation to Local, so that we can manipulate our airplane easily. Bear in mind however, that even if the manipulator is set to Local, Blender is using ZXY Euler internally to compute rotations and to interpolateangles, so what Blender is actually using is the ZXY gimbal. Now insert a rotation keyframe in frame 1, with the airplane in rest position (no rotations at all) as shown in figure 8.



Transform
Location: X: 0.000 Y: 0.000 Z: 0.000
Rotation: X: 0° Y: 0° Z: 0°
ZXY Euler
Scale: X: 1.000 Y: 1.000 Z: 1.000

Figure 8. Keyframe 1 (local axes shown).

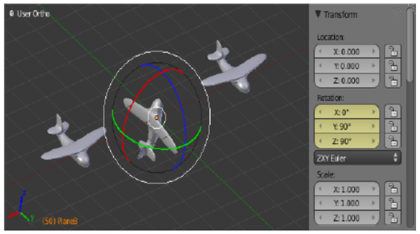
Now let's move to frame 25 using the arrow keys. In this frame, the pilot has astonished the audience of the airshow by setting the airplane in vertical position. Rotate 90º around the X local axis (remember that positive angles mean counterclockwise rotations), and set a new rotation keyframe (figure 9). Now the nose of the plane is pointing up. You can switch from Local to Global mode to see how the local axis has change. The "up" side of the plane is not the same as the "up" side of the world.



Transform
Location: X: 0.000 Y: 0.000 Z: 0.000
Rotation: X: 90° Y: 0° Z: 0°
ZXY Euler
Scale: X: 1.000 Y: 1.000 Z: 1.000

Figure 9. Keyframe 2 (local axes shown).

OK. But the pilot, who is a really bold guy, hasn't had enough. He wants to make a nice turn to his right while keeping the aircraft nose up. So now, get back to Local mode and go to frame 50. Then use the manipulator to rotate the airplane 90º around the Y local axis. Set a new rotation keyframe (figure 10).



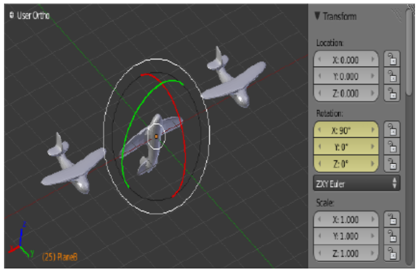
Transform
Location: X: 0.000 Y: 0.000 Z: 0.000
Rotation: X: 0° Y: 90° Z: 90°
ZXY Euler
Scale: X: 1.000 Y: 1.000 Z: 1.000

Figure 10. Keyframe 3 (local axes shown).

Now rewind to frame 1, and check the full animation using the arrow keys back and forth. You'll see that from keyframe 1 (frame 1) to keyframe 2 (frame 25) everything works as expected. But something weird happens between keyframe 2 and keyframe 3 (frame 50). We expect a right turn, but the airplane nose actually makes a weird movement.

To check what the problem was, set Gimbal orientation, rewind to frame 1, and check the animation again. As you approach frame 25, the Z rotation axis of the manipulator gets closer and closer to Y axis. At frame 25, the Z axis is completely aligned with the Y axis, as shown in figure 11.

We have just lost one axis of movement. This phenomenon is known as the Gimbal lock, and it's a very typical source of headache for animators.



Location:
X: 0.000
Y: 0.000
Z: 0.000
Rotation:
X: 90°
Y: 0°
Z: 0°
ZXY Euler
Scale:
X: 1.000
Y: 1.000
Z: 1.000

Figure 11. Keyframe 2 (gimbal shown)
Where is the Z axis...? Perfect gimbal lock

You may have found that your animated rotations behave in a weird manner (it has indeed happened to me), and there is no way to fix them no matter what you do to avoid the problem. Well, it's more than likely that you have been a victim of the hideous gimbal lock (ock... ock... ock...).

So, back to our airplane. We are nose up, and we have lost an axis to perform a right turn. For doing that kind of turn, we would need to get our Z axis back. Well, if you check the interpolation values of the animation between keys 2 and 3, you will realize that this is exactly what Blender does. While the Y axis rotates the commanded 90º, it also undoes the initial 90º rotation in the X axis (which caused the gimbal lock), and rotates 90º around the Z axis so that the final position can be reached.

The end result is the weird movement of the airplane. Unfortunately, that caused the audience to go back home, and the airshow resulted in a complete failure. Let's try to see when gimbal lock occurs, taking into account the type of Euler rotation we choose, so that we can avoid it.

We know there are three rotation axes in a gimbal. When all three axes are perpendicular between them, all is fine. However, as one of the axes starts to move towards another, they lose their relative perpendicularity, meaning that we are starting to lose some degree of freedom of movement. The problem reaches its maximum when two axes become completely aligned (parallel), that is, when we completely lose one of the three axes.

Let's take for instance a XYZ Euler gimbal. What happens when the axis at the bottom of the hierarchy (in this case, X) rotates? Nothing important actually. All three axes keep perpendicular whatever rotation you apply to the X axis, which just keeps spinning around itself.

What if we rotate the topmost axis in the hierarchy (Z)? Then all the axes in the system rotate with it, keeping their relative positions, without losing freedom of movement like before. The problem comes when we rotate the axis in the middle (Y). Its effect is to get its child axis (X) closer to its parent axis (Z). That said, one important thing to remember is that the middle axis in the Euler hierarchy is crucial, and we need to keep an eye on it most of all.

Now that we know when gimbal lock is reached, we can see how to avoid it. So, if you need an object to perform an animation with a series of rotations in which its Z axis will reach angles close to 90º (or equivalent angles like -90º or 270º), we will avoid the use of Euler rotation systems XZY and YZX, as in these, Z axis lays in the middle of the hierarchy.

However, we could still use XZY Euler even if the Z axis reaches 90º, but only if in those particular moments we don't need the X axis to rotate. We need to make sure that as soon as we need rotations around the X axis, Z rotation is far from 90º (and equivalents).

If you want to perform the former aerobatics, you can choose a different Euler system. For instance, you can repeat the experiment with a XYZ Euler system, and you will see everything working fine.

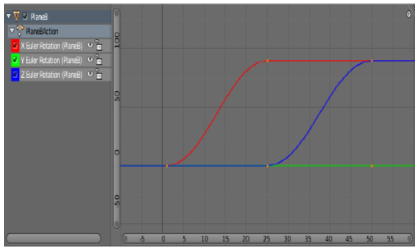


Figure 12. Euler rotations: animation curves.

When you are done with it, you can take a look at the resulting animation curves in the Graph editor (figure 12). See how intuitive those f-curves are. You can see a 90º rotation around the X axis take place between frame 1 and 25, and another 90º rotation around the Z axis between frames 25 and 50.

This is one of the big advantages of the Euler rotations as you can directly manipulate the rotation f-curves easily, knowing that the curves are independent between them. Those three curves give you a clear picture of what's going on, even if you don't see the actual object.

In this case all you have to do is keep an eye on the green curve (Y axis) and make sure it doesn't approach 90º when the red curve (X axis) is different than zero.

OK, but is there any rotation system which doesn't suffer from gimbal lock...?

Sure there is.

Axis Angle rotations

If you set the rotation mode to Axis Angle, you will notice that you now have 4 values for defining rotations: X, Y, Z and W.

With Euler we had 3 values representing a rotation angle around each axis. With axis-angle we define two things: one axis and one angle. The axis is defined by X, Y and Z; the rotation angle, by W. You can see that in figure 13.

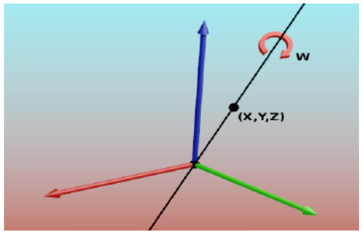


Figure 13. Axis-angle rotation.

The effective rotation is done around the axis (X,Y,Z). This axis is an infinite line that goes through the centre of the object and the point defined by (X,Y,Z) in the local coordinate system of the object. There are many ways to define the same axis. The most important thing is the ratio between these three values. Thus, (1,0.5,3) is the same axis as (2,1,6).

So once we have this rotation axis, all we have to do is to make the object rotate around it by the amount given in the W value. So if W=0, no rotation is applied, regardless of the values in X, Y and Z. Conversely, if X, Y and Z equal 0, no axis is defined, so once more, there will be no rotation regardless of the W value.

You can easily see that the most obvious advantage of axis-angle is rotation around an arbitrary axis. This makes axis-angle very suitable for objects that spin constantly around the same axis. The rotation of Earth around its peculiar axis is a perfect example.

Bear in mind that negative values matter in the axis definition, as axes have a direction. Two axes defined with opposite direction (simply changing the signs of X, Y, and Z) will have an opposite direction, and so, an inverse rotation. So changing the sign of all four values (axis and angle) has no effect on the final orientation (except for animation purposes).

One thing to note is that angle values in W are expressed in radians, not degrees. In that case, if you want to perform a 90º turn, you must rotate Π/2 in the W slider control. It's possible to type pi directly in the sliders as Blender knows its value (around 3.141592). You just need to know that 360 degrees are equivalent to 2xΠ radians, so that you can calculate other angles. You can directly enter values like pi/2, 3*pi/2, 2*pi, pi, etc. Anyway, manipulators and hotkey R always use degrees.

OK, now that we know what axis-angle is, it's time to play with it.

You can repeat the aerobatics experiment from scratch. Set rotation mode to Axis Angle, and redo the three keyframes using the 3D manipulator of choice, or hotkey **R**. Play the animation back.

What happens? Nothing good really.

Axis-angle rotation problems

As mentioned before, axis-angle works well for rotations around a fixed axis. So our aerobatics is not the best example to use. Let's see why it didn't work out smoothly (by now the pilot is depressed and already thinking about retiring).

What happened here is that from keyframe 1 to keyframe 2, two things were interpolated. First we went from axis (0,1,0) to axis (1,0,0). This axis movement is quite big, as it's going from one line to a perpendicular one (90º away). And second, we went from angle 0 to angle Π/2 (90º). So two things were moving the same amount: the axis and the angle.

Once again, let's go back to frame 1. Instead of axis (0,1,0), let's enter (1,0,0). It makes no difference, as the rotation angle value W is 0. Now update the keyframe and see how it goes.

Everything runs quite fine now. The second half is not perfect, but quite acceptable. Why isn't it perfect? It's important that between two consecutive keyframes most of the movement is taken by only one of the components: either the axis or the angle. In our initial airplane movement, both components were moving the same amount (90º), and that created a turbulent movement that made the pilot sick. Then we completely fixed the problem by keeping the axis still between keyframes. In the second half of the animation, the angle moves more than the axis, which is good, but both of them move.

If you want absolutely perfect movements, just move only one of the two components between two consecutive keyframes (usually the angle). Sometimes this is difficult, so the best thing in those situations is to start considering any other rotation system.

In axis-angle, rotation manipulators have to be used with special care, as they can lead to unwanted results. A simple rotation using the manipulator can lead, for instance, to the flipping (sign change) of the axis. If the initial axis is (0,1,0) and the final one results in (0,-1,0), that will produce, most probably, undesired effects, as we are changing its direction, which means a 180º rotation of the axis (not around the axis).

Moreover, in axis-angle you can define axes and angles using any value, as big or small as you want. Rotations can consist of several spins around the axis, that is, Nx2Π, where N is any number of revolutions, positive or negative. However, when using rotation manipulators you will always get axis values up to 1.0, and angles up to 2Π.

This, along with the possible axis flip, are good reasons to prefer editing rotation values directly on the Transform panel over using rotation manipulators or hotkey R.

To summarize, axis-angle is good for rotations around an arbitrary axis, as long as that axis doesn't keep moving, or at least its movement is really controlled. Remember that you can quickly move the axis to any value when rotation angle is 0 so that moment can be used to switch from one axis to another.

Now that you are done with the new air-show animation using axis-angle, take a look at the resulting animation curves (figure 14)... What can you see?

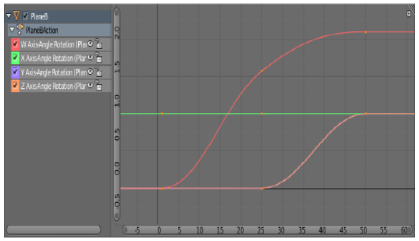


Figure 14. Axis-angle rotations: animation curves.

Yeah, right. Just curves. It's actually very difficult to know how they translate visually. While with Eulers we could grasp the meaning of the F-curves, now it's difficult to tell how the object is rotated.

Wouldn't it be nice, however, to have a rotation system which, while keeping its immunity to gimbal lock, at the same time produced perfect and smooth rotation interpolations, and not just fixed-axis ones?

Yeah, that would be awesome...!

Quaternion rotations

Quaternions were discovered by the Irish mathematician Sir William Rowan Hamilton.

According to the Wikipedia, "the breakthrough finally came on Monday 16 October 1843 in Dublin, when Hamilton was on his way to the Royal Irish Academy where he was going to preside at a council meeting. While walking along the towpath of the Royal Canal with his wife, the concept behind quaternions was taking shape in his mind. Hamilton could not resist the impulse to carve the formulae for the quaternions

$i^2 = j^2 = k^2 = ijk = -1$

into the stone of Brougham Bridge as he passed by it."

This reminds me of the day I was walking along the streets of my home town, and it came to my mind a recipe of beans with mushroom sauce. Immediately I took my chisel and hammer (I always bring them in my pockets, just in case). I couldn't resist carving the recipe into a stone of my neighbour's wall... Surprisingly, Wikipedia didn't mention that. My neighbour however, did mention it to his lawyer (he is allergic to mushrooms).

Back to quaternions, you can just forget about the formulae that Hamilton carved. Actually, you can forget about most of the maths around quaternions (unless you are a mathematician, a 3D software developer, or just very interested in Algebra).

A quaternion is a vector, i. e. a set of numbers, in a specific 4-dimensional space. In this case, this vector has four numbers. These four numbers are called X, Y, Z and W in Blender. Just check it in the Transform panel, setting Quaternion (WXYZ) rotation mode... Doesn't it remind you of something?

Of course, axis-angle has the same component names. So are these values related somehow to the corresponding axis-angle values? Absolutely. Let's see the differences, though.

In the first place, a quaternion can represent a rotation only if it is normalized, which means that the length (or modulus) of the vector must be 1 (this is called a unit vector). What does it mean in practice? Mathematically:

$$W^2 + X^2 + Y^2 + Z^2 = 1$$

This formula is not too useful for 3D artists. However it might help to understand how these four values relate to each other. In the first place, none can have an absolute value greater than 1. Second, when one value increases (in absolute value), the rest decrease, and vice versa. Absolute value means to forget about the sign, i.e., the absolute value of -0.75 is 0.75. So, all four values range from -1.0 to 1.0.

Now we know how quaternion values relate and affect each other. But what do they actually mean? Do they have the same meaning as in axis-angle? Well, actually they do. In a quaternion, X, Y and Z are still defining the same axis of rotation that axis-angle does, and W is defining an angle of rotation around that axis.

There is one unique (normalized) way to define a given rotation using a quaternion. On the other hand, in axis-angle you could define the same axis using many different combination of values, as the vector representing that axis didn't have to be normalized. Bear in mind though, that even in Axis Angle mode the rotation manipulator and the R hotkey also normalize the (X,Y,Z) vector.

Another question arises here; what units is W using to describe an angle, as it can only range from -1.0 to 1.0? To understand the correspondence between the axis-angle W value and the quaternion W value, we will call the first AW, and the second QW. Its relation is as follows:

$$QW = \cos(AW / 2)$$

If you know what a cosine function is, great. If not, don't worry the slightest bit. The only thing you should be aware of is how quaternion W behaves in relation to axis-angle W (the actual angle of rotation around the axis). The following table has a few examples that might help you:

Quaternion W	Angle in radians	Angle in degrees
1.000	0	0
0.707	$\pi / 2$	90
0.000	π	180
-0.707	$3 \times \pi / 2$	270
-1.000	$2 \times \pi$	360

You could think after seeing this table, that if a quaternion with W=1 is equivalent to a 0º angle, and with W=0 it represents 180º, then 90º should correspond to W=0.5. Actually it doesn't work like this, as you can see in the table, as the cosine doesn't behave like a linear function.

It actually behaves in a more "circular" way, which is much more suitable for rotations. Observing the table, you can think there is no way to use a quaternion to define a rotation beyond 360º, or below 0º. You think well. This is a small drawback of quaternions, but later we will see how to overcome it.

So let's try to see how a quaternion works. Open 'RotationsWorld.blend' once more and select one of the airplanes. Set rotation mode to Quaternion (WXYZ). Initially, W has value 1.0, which means a rotation of 0º, so we don't need any rotation axis. It doesn't matter then if X, Y and Z are all zero.

Now increase the value of X slightly, clicking on the right triangle in the X slider of the Transform panel. See what happens. We have just defined an axis; a point in the direction of the X axis defines the X axis itself. If we keep increasing the X value, we are still defining the same axis, however W decreases. The bigger the value of X, the smaller the value of W. In other words, we are rotating around X axis, as W keeps decreasing towards 0, i. e. towards 180º (see the table). When X reaches 1, W is 0, which means a rotation of 180º around the X axis.

So the effect of increasing the value of X is to bring the object to this position; upside down around the X axis. Now clear the rotation. You can repeat the same experiment with Y and Z values. As you will see, all of them try to bring the object upside down around their own axis.

On the other hand, what is the effect of making W bigger? Obviously to take the object away from those upside down positions, and preserve the original position with no rotations at all. <u>The balance between the four values is what defines the final rotation</u>.

If you repeat the experiment using negative values, you will see the same effect but in the opposite rotation direction. Take for instance the experiment around the X axis, but this time taking it slowly towards -1.0. We are defining the same rotation values (W is still positive) but applied around an axis that runs along the X axis in the opposite direction. This is similar to what happened with axis-angle. In this case also, changing the sign of all four values has no effect on the final rotation. And with quaternions, it doesn't have an effect on animation interpolations either.

Even if in theory W cannot hold a number corresponding to a negative angle, changing its sign works in a similar fashion. For instance, W=0.707 represents a 90º rotation, while W=-0.707 is 270º, which is in fact equivalent to -90º (270º=360º-90º).

Now that we know what a quaternion is and how it works, we are ready to repeat the airshow. Set the three keyframes once more using Quaternion (WXYZ) mode. What happens now?

An incredible aerobatic manoeuvre. The audience is shouting, jumping, hugging, laughing...! The best show ever! And all thanks to Sir Hamilton and his magic chisel...

What about quaternion animation curves? Take a look at them (figure 15)... What do you think? Yeah, awful. Forget about animating those evil f-curves... And there is more. In quaternion f-curves, Linear Extrapolation doesn't work well. Since the quaternion must be normalized, its values can't keep growing forever. The relation between the four values ends up reaching a normalized balance, and so the rotation slowly stops at that point.

You have seen the main advantage of quaternions: its absolute smoothness and perfection in interpolations, no gimbal lock, no weird movements, etc. However, we have seen a drawback: the inability to define more than one revolution, or negative angles. Let's see an example. Open a Blender scene and take any unrotated object (our good old airplane will do).

Choose Quaternion (WXYZ) rotation mode. Set a keyframe in frame 1, without rotation. Go to frame 25. Rotate it 200º counterclockwise around Z axis. Set the rotation key. Check the animation. What happens? Blender actually interpolates using a clockwise rotation!

Blender has chosen the shortest path between 0º and 200º, which is equivalent to -160º. Quaternions can't define revolutions (successive spins around an axis). They just define orientations in space; from 0º to 360º, where 0º is equivalent to 360º (value 1 is equivalent to -1 for X, Y, Z and W parameters). Blender always performs the shortest path rotation between one orientation and the next if you use the rotation manipulators or hotkey R. This means you can't rotate 180º or more between two keyframes using those. But you can get bigger angles by directly editing the Transform panel values. However you will never get a 360º or bigger rotation.

If you want to overcome this, and make the object spin several times, you have to set intermediate keyframes between the initial and final state, so that turns between them are smaller than 360º (or -360º for clockwise rotation). If you use manipulators, turns must be less than 180º (or -180º).

Gimbals and locks

No, we are not going to talk about gimbal lock anymore. Just about gimbals, and component locks in either quaternion and axis-angle rotations.

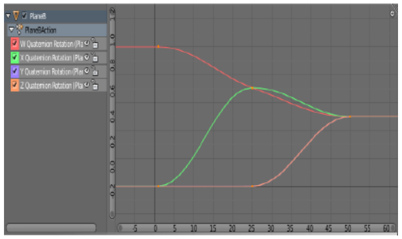

PlaneB
PlaneBAction
W Quaternion Rotation (Pla
X Quaternion Rotation (Pla
Y Quaternion Rotation (Pla
Z Quaternion Rotation (Pla

Figure 15. Quaternion rotations: animation curves.

Provided that neither of these two systems use Euler gimbals, what is the meaning of the Gimbal orientation of the 3D manipulators?

In Axis Angle, you will see that Gimbal aligns its Z component with the defined axis (X,Y,Z), so that if you rotate the manipulator blue ring (Z axis) you will be directly controlling the W value, and just the W value. However, if you want negative values, or values beyond 2π, you must edit the W value in the Transform panel.

On the other hand, when using quaternions you can see that Gimbal currently has no special meaning and is equivalent to the Local orientation. Perhaps future releases of Blender will give it a special use.

Regarding the lock buttons in the Transform panel, their use is to restrict rotations (and locations/scaling) to only the desired axes using the 3D manipulators or hotkey **R**. However, in the specific case of rotations, if you activate the **4L** button, you can restrict rotations by axis-angle or quaternion component instead of axis.

As they have 4 components (X, Y, Z, W), you get an extra lock. However, in the specific case of quaternions, remember that even changing just one of the components will affect the other three as the final vector must be normalized.

Summary

- Local or global rotation systems aren't valid for computing rotations as the order of rotation around the three axes affects the final result.
- Euler rotation systems use a hierarchy of rotation axes which is valid to compute rotations, as the three rotation components are independent. However they suffer from gimbal lock in certain circumstances.
- Axis-angle doesn't suffer from gimbal lock, but its use is almost specific to revolving around a fixed axis.
- Quaternion system doesn't suffer from gimbal lock, and interpolates perfectly any pair of orientations. However it can't define successive revolutions unless we insert intermediate keyframes in between. As it's a perfect way to define orientations in space, it is very suitable for bones animation.
- Regarding animation curves, the Euler system is the only one that provides an easy and intuitive way to edit them.

And finally...

There are a couple videos around there that might help you see rotations in action. Check the Guerrilla CG Project website (guerrillacg.org). Watch the following videos: The Rotation Problem, and Euler Rotations Explained. One warning, though: in the first of these videos there is a small mistake; whenever 'Quaternion' is mentioned, it should actually say 'Axis Angle'.

There are other 3D software packages out there that use other rotation systems, like the Heading/Pitch/Bank (or Yaw/Pitch/Roll) angles, used with the so called Tait-Bryan or cardan angles, which are a different kind of Euler rotations. But this stuff is out of the scope of this article as Blender doesn't use them ■

I hope not to have made your head rotate too much.

Be good!