HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
Fakultät für Informatik
Labor für Computergrafik
Prof. Dr. G. Umlauf

Konstanz, 08.10.2014

Aufgabe 1 & 2 & 3

„Geometrisches Modellieren“

Besprechung und Abgabe spätestens am 12.11.2014, F031/F033.

Vorbemerkung:
Benutzen Sie für diese Übung **keine** OpenGL, GLUT oder GLAUX Funktionen, die Projektionen und Rotationen berechnen! Benutzen Sie die zur Verfügung gestellten Vektor und Matrix Klassen.

Programmgerüst für die Aufgabe:
Laden Sie sich die zip-Datei zur Übung von der web-Seite der Vorlesung:

- Darin enthalten ist ein VC-Projekt Version 2010.
- Darin enthalten sind weiter:
  - `main.cpp`:
    Programmgerüst, das die Benutzung einer Display-Funktion (zum Refresh des Doublebuffer) und eine Keyboard-Funktion zur Steuerung enthält.
  - `color.h, vec.h, mat.h`:
    Implementierungen einer Farb-, Vektor- und Matrixklasse. Einige Funktionen sind bereits exemplarisch implementiert. Implementieren Sie bei Bedarf die fehlenden Funktionen. Orientieren Sie sich dabei an dem existierenden Code.
  - `viewSystem.h, viewSystem.cpp`:
    Implementierungen des Sichtsystems, bestehend aus Augpunkt (`EyePoint`), Blickrichtung (`ViewDir`) und Bildebene (`ViewUp, ViewHor`) in homogenen Koordinaten. Das Sichtsystem übernimmt die 3d-2d-Projektione, globale Koordinatensystem-Transformationen sowie affine Abbildungen. Sie verfügt über drei Modi zur Auswahl, welche Implementierung der affinen Abbildungen verwendet werden soll:
    - `VIEW_MATRIX_MODE, VIEW_FORMULA_MODE:` Implementierung mit 4x4 Matrizen (funktionsfähig).
    - `VIEW_QUATERNION_MODE:` Implementierung mit Quaternionen (siehe Aufgabe 02).
  - `quader.h, quader.cpp`:
    Implementierungen einer Klasse zur Repräsentation von Quader-Objekten.
  - `quaternion.h, quaternion.cpp`:
    Implementierungen einer Klasse zur Repräsentation und Verwendung von Quaternionen.

Die Funktionalität aus beiden Aufgaben wird anhand des Source-Codes überprüft!

Aufgabe 01 (Allgemeine Orientierung berechnen)

In dem Programmgerüst kann die Szene nur aus der Ansicht des Sichtsystem (`ViewSystem`) betrachtet werden, das beliebig im Raum positioniert sein kann. Gegeben seien also das Sicht-Koordinatensystem:

- Eine allgemeine Position des Augpunktes (in homogenen Koordinaten) `EyePoint`.
- Eine allgemeine Sichtrichtung `ViewDir`.
- Ein allgemeiner View-Up-Vektor `ViewUp`.
- Die zusätzliche (implizierte) Richtung `ViewHor = ViewDir x ViewUp`.

Implementieren im Sie in `viewSystem.cpp` die Methoden

```
          CMat4f getTransform?(),
```

die die Lage des Sicht-Koordinatensystem als 4×4-Matrix zur Transformation in das Welt-Koordinatensystem berechnet (siehe Abbildung 1). Diese führt auch auf eine Matrix, die Punkte aus dem Welt-Koordinatensystem in das Sicht-Koordinatensystem überführt.

Wenn Sie diese Methoden implementiert haben, sollte die Szene mit den Tastenkürzeln aus Aufgabe02 im `VIEW_FORMULA_MODE` (in `main.cpp` setzen) manipulierbar sein.

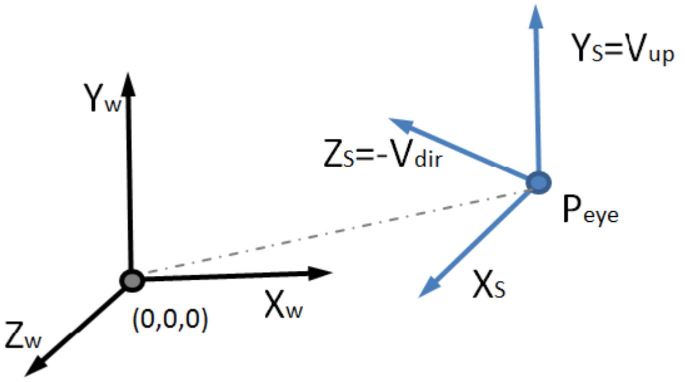


Abbildung 1 Allgemeine Sichttransformation.

Aufgabe 02 (Quaternionen)

Schreiben Sie nun ein Programm so um, dass es im `VIEW_FORMULA_MODE` verwendet werden kann. Erstellen Sie dazu für die Keyboard-Interaktion die fehlenden Methoden und Operationen im `viewSystem.cpp` und `quaternion.cpp`:

a. `X`, `Y` und `Z` rotieren das Sicht-Koordinatensystem in positiver Richtung um die x-, y- und z-Achse des Welt-Koordinatensystems und `x`, `y` und `z` drehen in negativer Richtung um die jeweilige Achse.

b. `A`, `B` und `C` bzw. `a`, `b` und `c` rotieren das Sicht-Koordinatensystem in entsprechender Richtung um die jeweilige Achse des Sicht-Koordinatensystems.: `A,a`: ViewDir-Vektor, `B,b`: ViewUp-Vektor, ~~`B,b`~~: ViewHor-Vektor.

c. `U`, `V`, `W`, `u`, `v` und `w` verschieben das Sicht-Koordinatensystem entlang der Achsen des Welt-Koordinatensystems in die entsprechenden Richtungen: `U,u`: x-Achse, `V,v`: y-Achse, `W,w`: z-Achse (nicht Teil der Aufgabe).

d. `R` setzt das Sicht-Koordinatensystem in den Anfangszustand (deckungsgleich mit dem Welt-Koordinatensystem) zurück (nicht Teil der Aufgabe).

e. `f` und `F` ändert den Fokalabstand des Sichtsystems (nicht Teil der Aufgabe).

Zum Zeitpunkt der Initialisierung der Applikation sollen Welt-Koordinatensystem und Sicht-Koordinatensystem deckungsgleich sein.

Aufgabe 03 (Rotationsinterpolation)

Implementieren Sie in Ihrem Programm drei Methoden zur Interpolation von Rotationen. Wählen Sie dazu eine Start- und eine Endposition Ihrer Szene und interpolieren Sie zwischen diesen beiden Positionen entsprechende Zwischenpositionen mit den folgenden drei Methoden, siehe [Shoemaker 1985] und [Kremer 2008].

a. Lineare Interpolation (LERP) zwischen den beiden Positionen.

b. Spherical linear interpolation (SLERP) zwischen den beiden Positionen.

c. Normalized SLERPs (NLERP) zwischen beiden Positionen.

Besprechung und Abgabe spätestens am 12.11.2014, F031/F033.

Animating Rotation with Quaternion Curves

Ken Shoemake[†]

The Singer Company
Link Flight Simulation Division

ABSTRACT

Solid bodies roll and tumble through space. In computer animation, so do cameras. The rotations of these objects are best described using a four coordinate system, quaternions, as is shown in this paper. Of all quaternions, those on the unit sphere are most suitable for animation, but the question of how to construct curves on spheres has not been much explored. This paper gives one answer by presenting a new kind of spline curve, created on a sphere, suitable for smoothly in-betweening (i.e. interpolating) sequences of arbitrary rotations. Both theory and experiment show that the motion generated is smooth and natural, without quirks found in earlier methods.

C.R. Classification: G.1.1 [Numerical Analysis] Interpolation—Spline and piecewise polynomial interpolation; G.1.2 [Numerical Analysis] Approximation—Spline and piecewise polynomial approximation; I.2.9 [Artificial Intelligence] Robotics—Manipulators; I.3.5 [Computer Graphics] Computational Geometry and Object Modelling—Curve, surface, solid, and object representation, —Geometric algorithms, languages, and systems, —Hierarchy and geometric transformations

General Terms: Algorithms, Theory

Keywords and phrases: quaternion, rotation, spherical geometry, spline, Bézier curve, B-spline, animation, interpolation, approximation, in-betweening

1. Introduction

Computer animation of three dimensional objects imitates the *key frame* techniques of traditional animation, using key positions in space instead of key drawings. Physics says that the general position of a rigid body can be given by combining a translation with a rotation. Computer animators key such transformations to control both simulated cameras and objects to be rendered. In following such an approach, one is naturally led to ask: What is the best representation for general rotations, and how does one in-between them? Surprisingly little has been published on these topics, and the answers are not trivial.

This paper suggests that the common solution, using three Euler's angles interpolated independently, is not ideal. The more recent (1843) notation of quaternions is proposed instead, along with interpolation on the quaternion unit sphere. Although quaternions are less familiar, conversion to quaternions and generation of in-between frames can be completely automatic, no matter how key frames were originally specified, so users don't need to know—or care—about inner details. The same cannot be said for Euler's angles, which are more difficult to use.

Spherical interpolation itself can be used for purposes besides animating rotations. For example, the set of all possible directions in space forms a sphere, the so-called Gaussian sphere, on which one might want to control the positions of infinitely distant light sources. Modelling features on a globe is another possible application.

It is simple to use and to program the method proposed here. It is more difficult to follow its development. This stems from two causes: 1) rotations in space are more confusing than one might think, and 2) interpolating on a sphere is trickier than interpolating in, say, a plane. Readers well acquainted with splines and their use in computer animation should have little difficulty, although even they may stumble a bit over quaternions.

2. Describing rotations

2.1 Rigid motion

Imagine hurling a brick towards a plate glass window. As the brick flies closer and closer, a nearby physicist

[†] Author's current address: 1700 Santa Cruz Ave., Menlo Park, CA 94025

might observe that, while it does not change shape or size, it can tumble freely. Leonhard Euler proved two centuries ago that, however the brick tumbles, each position can be achieved by a single rotation from a reference position. [Euler,1752] [Goldstein] The same is true for any rigid body. (Shattering glass is obviously not a single rigid body.)

While translations are well animated by using vectors, rotation animation can be improved by using the progenitor of vectors, quaternions. Quaternions were discovered by Sir William Rowan Hamilton in October of 1843. The moment is well recorded, for he considered them his most important contribution, the inspired answer to a fifteen-year search for a successor to complex numbers. [Hamilton] By an odd quirk of mathematics, only systems of two, four, or eight components will multiply as Hamilton desired; triples had been his stumbling block.

Soon after quaternions were introduced, Arthur Cayley published a way to describe rotations using the new multiplication. [Cayley] The notation in his paper so closely anticipates matrix notation, which he devised several years later, that it may be taken as a formula for converting a quaternion to a rotation matrix. It turns out that the four values making up a quaternion describe rotation in a natural way: three of them give the coordinates for the axis of rotation, while the fourth is determined by the angle rotated through. [Courant & Hilbert]

Since computer graphics leans heavily on vector operations, it is perhaps easiest to explain quaternions and rotation matrices in terms of these, reversing history. However quaternions can stand on their own as an elegant algebra of space. [Herstein] [Pickert] [MacLane]

2.2 Rotation matrices

That a tumbling brick does not change size, shape, nor "handedness" is mathematically expressed as the preservation of dot products and cross products, since these measure lengths, angles, and handedness. And since the determinant of a 3×3 matrix can be computed as the dot product of one column with the cross product of the other two, determinants are also preserved. Symbolically:

$$\mathrm{Rot}(\underline{v}_1)\cdot\mathrm{Rot}(\underline{v}_2) = \underline{v}_1\cdot\underline{v}_2$$

$$\mathrm{Rot}(\underline{v}_1)\times\mathrm{Rot}(\underline{v}_2) = \mathrm{Rot}(\underline{v}_1\times\underline{v}_2)$$

$$\det(\mathrm{Rot}(\underline{v}_1),\mathrm{Rot}(\underline{v}_2),\mathrm{Rot}(\underline{v}_3)) = \det(\underline{v}_1,\underline{v}_2,\underline{v}_3)$$

An immediate consequence is that orientation changes must be linear operations, since the preserved operations are; hence they have a matrix representation, M. Using the matrix form of a dot product, $\underline{v}_1^t\ \underline{v}_2$, we can say more precisely that $(M\ \underline{v}_1)^t\ (M\ \underline{v}_2) = \underline{v}_1^t\ \underline{v}_2$, from which it follows that

$$M^t\ M = I\ .$$

That is, the change matrix M is *orthogonal*; its columns (and rows) are mutually perpendicular unit magnitude vectors. Because M must also preserve determinants, it is a *special orthogonal* matrix, satisfying

$$\det(M) = +1\ .$$

It is well known, and anyhow easy to show, that the special orthogonal matrices form a group, $SO(3)$, under multiplication. [MacLane][Goldstein][Misner] In this rotation group, the inverse of M is just M^t, the opposite rotation.

To illustrate, the matrix

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

effects a rotation through an angle of θ around the x axis. After verifying the properties discussed so far, note that the diagonal entries sum to $1+2\cos\theta$. While it is too lengthy to show here, the diagonal sum measures the same quantity for matrices generating rotation around any axis. [MacLane]

2.3 Quaternions

Quaternions, like rotations, also form a non-commutative group under their multiplication, and these two groups are closely related. [Goldstein] [Pickert][Misner] In fact, we can substitute quaternion multiplication for rotation matrix multiplication, and do less computing as a result. [Taylor]

To perform quaternion arithmetic, group the four components into a real part—a scalar, and an imaginary part—a vector. Addition is easy: add scalar to scalar and vector to vector. But our major interest is in multiplication. Start with a simple case: multiply two quaternions without real parts, or more precisely, with zero real parts. The result quaternion has a vector that is the cross product of the two vector parts, and a scalar that is their dot product, negated:

$$\underline{v}_1\underline{v}_2 = [(-\underline{v}_1\cdot\underline{v}_2),(\underline{v}_1\times\underline{v}_2)]\ .$$

It is certainly convenient to encompass both vector products with a single quaternion product. (One early lover of quaternion algebra called vector algebra a "hermaphrodite monster", since it required two kinds of product, each yielding a different type of result.) If one quaternion has only a scalar part, with its vector components all zero, multiplication is just real multiplication and vector scaling. Combining the two effects gives the general rule [Brady]:

$$[s_1,\underline{v}_1]\ [s_2,\underline{v}_2] = [(s_1s_2-\underline{v}_1\cdot\underline{v}_2),(s_1\underline{v}_2+s_2\underline{v}_1+\underline{v}_1\times\underline{v}_2)]\ .$$

Except for the cross product this looks like complex multiplication, $(a_1+\mathrm{i}b_1)(a_2+\mathrm{i}b_2) = (a_1a_2-b_1b_2) + \mathrm{i}(a_1b_2+a_2b_1)$, as Hamilton intended.†

Quaternions multiply with a cross product because rotations confound axes. To illustrate , place a book in front of you, face up, with the top farthest away. Use this orientation as a reference. Now hold the sides and flip it toward you onto its face, rotating 180 degrees around a left-to-right axis, y. Then, keeping it face down, spin it clockwise 180 degrees around an up-down z axis. Two rotations around two perpendicular axes; yet the total change in orientation must be, according to Euler, a single rotation. Indeed, if you hold the ends of the spine and flip the book 180 degrees around this third, outward-pointing, x axis, you should restore the original orientation. As quaternions, this is —anticipating developments ahead— [0,(0,1,0)] times [0,(0,0,1)] equals [0,(1,0,0)]; the cross product is essential.

Notice how quaternion operations give a new orientation, in "quaternion coordinates", much as translations give a position, relative to some starting reference. A central message of this paper is that quaternion coordinates are best for interpolating orientations. For comparison, imagine using spherical coordinates for translations! Quaternions represent orientation as a single rotation, just as rectangular coordinates represent position as a single vector. Translations combine by adding vectors; rotations, by multiplying quaternions. The separate axes of translations don't interact; the axes of rotations must. Quaternions preserve this interdependence naturally; Euler's angle coordinates ignore it.

2.4 Euler's angles

Why, then, do so many animators use Euler's angles? Mostly, I suspect, because quaternions are unfamiliar. Unlike Euler's angles, quaternions are not taught early in standard math and physics curricula. Certainly there is a plethora of arguments against angle coordinates. Euler's angle coordinates specify orientation as a series of three independent rotations about pre-chosen axes. For example, the orientation of an airplane is sometimes given as "yaw" (or "heading") around a vertical axis, followed by "pitch" around a horizontal axis through the wings, followed by "roll" around the nose-to-tail line. These three angles must be used in exactly the order given because rotations do not commute. The ordering of rotation axes used is a matter of convention, as is the particular set of axes, no matter what the order. For instance some physicists use the body centered axes z-x-z, in contrast to the aeronautics z-y-x. At least a dozen different conventions are possible for which series of axes to use. [Kane][Goldstein] The geometry of orientations in Euler's angle coordinates is contorted, and varies with choice of initial coordinate axes. There is no reasonable way to "multiply" or otherwise combine two rotations. Even converting between rotation matrices and angle coordinates is difficult and expensive, involving arbitrary assumptions and trigonometric functions. In their defense, it must be said that they are handy for solving differential equations—which is how Euler used them. [Euler,1758]

3. In-betweening alternatives

3.1 Straight line in-betweening

It is not immediately obvious how to in-between even two rotation keys. What orientations should an object assume on its journey between them? A natural answer is: take the first key as a reference, and represent the second by describing the single rotation that takes you to it, according to Euler's theorem. The in-between orientations should be positioned along that rotation.

If we plot quaternions as points in four-dimensional space, the straight lines between them give orientations interpolating the end points in exactly the above sense. If we plot Euler's angle coordinates instead, the in-between orientations will try to twist around three different axes simultaneously. This angle interpolation treats the three angles of rotation at each key orientation as a three-dimensional vector whose components are interpolated independently from key to key. Paradoxically, we can not rotate simply except around the special axes chosen for composition. We may even encounter so-called "gimbal lock", the loss of one degree of rotational freedom. Gimbal lock results from trying to ignore the cross product interaction of rotations, which can align two of the three axes. Quaternions are safe from gimbal lock, and so have been used for years to handle spacecraft, where it is unacceptable. [Kane][Mitchell]

3.2 How quaternions rotate

Straight lines between quaternions, however, ignore some of the natural geometry of rotation space. If our interpolated points were evenly spaced along a line, the animated rotation would speed up in the middle. To see why, we must look at how a quaternion converts to a rotation matrix. We rotate a vector by a quaternion so: multiply it on the right by the quaternion and on the left by the inverse of the quaternion, treating the vector as $[0,\underline{v}]$.

$$\underline{v}' = \mathrm{Rot}(\underline{v}) = q^{-1}\ \underline{v}\ q$$

Though it is not obvious, the result will always be a vector, with a zero scalar component. Notice how this guarantees

$$\mathrm{Rot}(\underline{v}_1)\ \mathrm{Rot}(\underline{v}_2) = \mathrm{Rot}(\underline{v}_1\ \underline{v}_2)$$

which implies that dot and cross products are preserved, embedded in the quaternion product.

The inverse of a quaternion is obtained by negating its

† Hamilton wrote a quaternion as $s+\mathbf{i}v^x+\mathbf{j}v^y+\mathbf{k}v^z$, with $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. The multiplication rules given before are consequences of this elegant formulation.

vector part and dividing both parts by the magnitude squared. For $q = [s,\underline{v}]$,

$$q^{-1} = \frac{1}{||q||^2}[s,-\underline{v}] \; ; \qquad ||q||^2 = s^2+\underline{v}\cdot\underline{v} \; .$$

Because all effects of magnitude are divided out, any scalar multiple of a quaternion gives the same rotation. (This kind of behavior is not unknown in computer graphics; any scalar multiple of a point in homogeneous coordinates gives the same non-homogeneous point.)

If the scalar part has value w, and the vector part values x, y, and z, the corresponding matrix can be worked out to be

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

when the magnitude $w^2+x^2+y^2+z^2$ equals 1. The magnitude restriction implies that, plotted in four-dimensional space, these quaternions lie on a sphere of radius one. Deeper investigation shows that such unit quaternions carry the amount of rotation in w, as $\cos \theta/2$, while the vector part points along the rotation axis with magnitude $\sin \theta/2$. The axis of a rotation is that line in space which remains unmoved; but notice that's exactly what happens when scalar multiples of $\underline{v}$ are rotated by $[s,\underline{v}]$. Because the cross product drops out, multiplication commutes, q^{-1} meets q, mutual annihilation occurs, and the vector emerges unscathed. Summing the matrix diagonal leads to the formula stated for w. The sum equals $4w^2-1$, but must also be $1+2\cos\theta$. A trig identity, $\cos 2\theta = 2\cos^2\theta-1$, finishes the demonstation.

3.3 Great arc in-betweening

This sphere of unit quaternions forms a sub-group, S^3, of the quaternion group. Furthermore, the spherical metric of S^3 is the same as the angular metric of $SO(3)$. [Misner] From this it follows that we can rotate without speeding up by interpolating on the sphere. Simply plot the two given orientations on the sphere and draw the great circle arc between them. That arc is the curve where the sphere intersects a plane through the two points and the origin. We sped up before because we were cutting across instead of following the arc; otherwise the paths of rotation are the same.

A formula for *s*pherical *l*inear int*erp*olation from q_1 to q_2, with parameter u moving from 0 to 1, can be obtained two different ways. From the group structure we find

$$\mathrm{Slerp}(q_1,q_2;u) = q_1(q_1^{-1}q_2)^u \; ;$$

while from the 4-D geometry comes†

$$\mathrm{Slerp}(q_1,q_2;u) = \frac{\sin\,(1-u)\theta}{\sin\theta}\,q_1 + \frac{\sin\,u\theta}{\sin\theta}\,q_2 \, ,$$

where $q_1 \cdot q_2 = \cos\theta$. The first is simpler for analysis, while the second is more practical for applications.

But animations typically have more than two key poses to connect, and here even our spherical elaboration of simple linear interpolation shows flaws. While orientation changes seamlessly, the direction of rotation changes abruptly. In mathematical terms, we want higher order continuity. There are lots of ways to achieve it—off the sphere; unfortunately we've learned too much.

3.4 Rotation geometry and topology

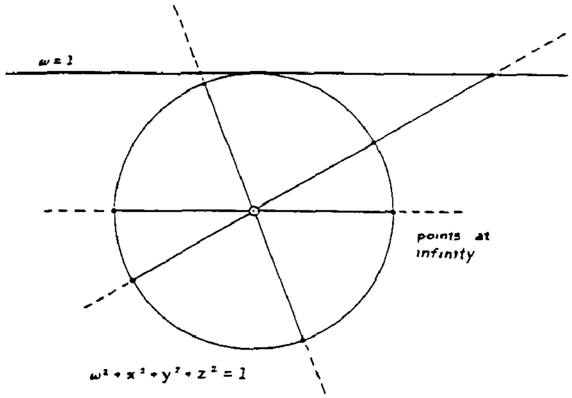
No matter what we do in general quaternion space, the ultimate effect must be interpreted via the sphere; so we had best work there in spite of the difficulty. It is important to grasp this point. The metric structure, hence the intrinsic geometry, of the rotation group $SO(3)$ is that of a sphere. Over small regions, meaning in this case small rotation angles, a sphere looks as if it is flat. But if we go far enough along a "straight line", we end up back where we started. What could be more evident about rotations? Their very essence is moving in circles. Looking back to the book-turning experiment, the confounding of axes is like traveling on a sphere: if we go in some direction to a quarter of the way around the sphere, turn 90 degrees, travel the same distance, then turn and travel again, we will arrive back home, coming in at right angles to the direction we headed out. Even more revealing, we can leave the north pole in any direction and end up at the south pole, just as we can rotate 360 degrees around any axis and end up oriented the same way.

Local geometry does not, however, determine global topology. Contradictory though it may seem, the geometry curves like a sphere, but the topology says north and south poles are the same! In fact, each pair of opposite points represents the same rotation. The reader may preserve sanity through two expedients. One is to see that this, like homogeneous coordinates, is geometry under perspective projection. The second is to restore spherical topology by including "entanglements". Physically, taking an object with strings attached and rotating it 360 degrees leaves the strings tangled; yet—most odd—rotating 720 degrees does not. [Misner][Gardner]

Accepting the topological oddity is more useful here, but it leaves a minor inconvenience. Namely, when converting an orientation in some foreign form, such as a matrix, to a quaternion form, which quaternion should we choose? Which side of the sphere? An answer that works well is this. Construct a string of quaternions through which to interpolate by choosing

† Glenn Davis suggested this formula.

each added quaternion on the side closest to the one before. Then small changes in orientation will yield small displacements on the sphere.



Representing a projective plane

3.5 Splines

We are left with the problem of constructing smooth curves on spheres. About a hundred years after quaternions appeared, Isaac Schoenberg published a two part attack on ballistics and actuarial problems, using what he called splines. [Schoenberg] Named by analogy to a draftman's tool, these are interpolating curves constructed from cubic polynomial pieces, with second order continuity between pieces. Cubic splines solve an integral equation which says to minimize the total "wiggle" of the curve, as measured by the second derivative. These interpolants are very popular, and the equation can be augmented with Lagrange multipliers to constrain the solution curves to lie on a sphere [Courant & Hilbert]; yet there are problems. First, the augmented equation is much more difficult and expensive to solve. Second, the curve must adjust everywhere if one of the points changes; that is, we have no local control.

3.6 Bézier curves

While Schoenberg invented splines based on numerical analysis, Pierre Bézier invented a class of curves, now called by his name, based on geometrical ideas. In fact, he showed how to find points on such a curve by drawing lines and splitting them in regular proportions. [Bézier] This is exactly what is needed. We already know how to do the equivalent—draw great arcs and proportions of arcs—on a sphere. A complete solution needs only a little more.

4. Spherical Bézier curves

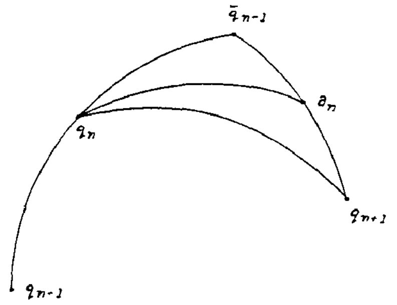
4.1 Joining curves

Bézier curves go through only their first and last defining points, but we want to interpolate all our orientations. The trick is to splice together short Bézier curves in the manner of splines. Their creator showed an easy way to do this which guarantees first order continuity, probably enough for us. As the curve goes through its end points it is tangent to its end segments. Line up the segments across a join, match their lengths, and the curves will piece together smoothly. If the key orientations are placed at joints, then each short curve moves us from one key to the next, because each piece passes through its ends.

Now, although the two segments abutting a curve junction should match each other, one of the segments can be chosen freely. These choices determine the axis and speed of rotation as we pass through the keys. The burden of choice can be passed to the animator of course, but automation is feasible, and generally preferable.

4.2 Choosing joint segments

Spherical linear interpolation gives two conflicting arc segments at a joint, one on each side. Smooth the difference with an even compromise, aiming for a point halfway between where the incoming segment would proceed, and where the outgoing segment must arrive.†



Constructing a point for tangent

Given successive key quaternions q_{n-1}, q_n, q_{n+1} interpretted as 4-D unit vectors, the computation for a segment point a_n after q_n is

$$a_n = \text{Bisect}(\text{Double}(q_{n-1}, q_n), q_{n+1}) ,$$

where

$$\text{Double}(p,q) = 2(p \cdot q)q - p ;$$

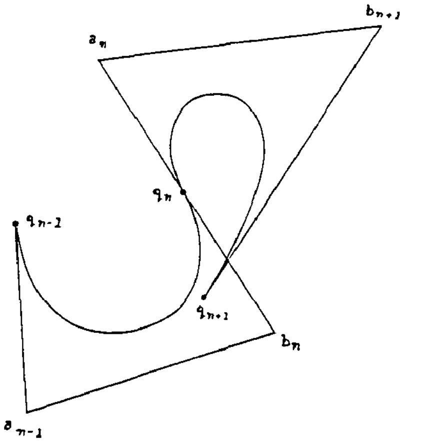
$$\text{Bisect}(p,q) = \frac{p+q}{||p+q||} .$$

The matching point for the segment before q_n should be

$$b_n = \text{Double}(a_n, q_n)$$

† For the numerically knowledgeable, this construction approximates the derivative at points of a sampled function by averaging the central differences of the sample sequence. [Dahlquist & Björk]

to ensure a smooth join, regardless of how a_n is chosen.



Splicing Bèzier segments together

4.3 Evaluating on the sphere

Everything is now in hand to imitate Bézier's curve technique. Each short curve is defined by four quaternions, q_n, a_n, b_{n+1}, q_{n+1}. Let the parameter u vary from 0 to 1 as the curve departs q_n towards a_n and arrives at q_{n+1} tangent to the arc from b_{n+1}. Spherically interpolate by proportion u between q_n and a_n, a_n and b_{n+1}, b_{n+1} and q_{n+1}, to obtain three new quaternions. Then interpolate between those to get two more; and finally interpolate again, reducing to a single point. Abbreviating Slerp$(p,q;u)$ as $(p:q)_u$, the computation looks like this:

$$q_n = p_0^{(0)}$$

$$(p_0^{(0)}:p_1^{(0)})_u = p_0^{(1)}$$

$$a_n = p_1^{(0)} \qquad (p_0^{(1)}:p_1^{(1)})_u = p_0^{(2)}$$

$$(p_1^{(0)}:p_2^{(0)})_u = p_1^{(1)} \qquad (p_0^{(2)}:p_1^{(2)})_u = p_0^{(3)} = q_{n+u}$$

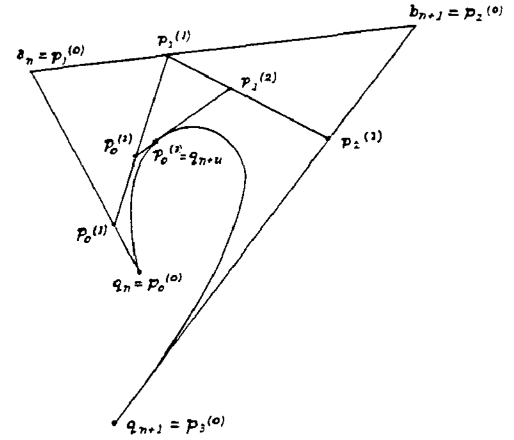
$$b_{n+1} = p_2^{(0)} \qquad (p_1^{(1)}:p_2^{(1)})_u = p_1^{(2)}$$

$$(p_2^{(0)}:p_3^{(0)})_u = p_2^{(1)}$$

$$q_{n+1} = p_3^{(0)}$$

4.4 Tangents revisited

A simple check proves the curve touches q_n and q_{n+1} at its ends. A rather challenging differentiation shows it is tangent there to the segments determined by a_n and b_{n+1}. However, as with Bézier's original curve, the magnitude of the tangent is three times that of the segment itself. That is, we are spinning three times faster than spherical interpolation along the arc. Fortunately we can correct the speed by merely truncating the end segments to one third their original length, so that a_n is closer to q_n and b_{n+1} closer to q_{n+1}.



Calculating a Bèzier curve point recursively

5. Results

5.1 The grand scheme

What have we ended up with? An animator sits at a workstation and interactively establishes a sequence of keys for, say, camera orientation. The interpolating algorithm does not depend on the nature of the interface the animator sees; all needed information is contained in the sequence of keys. Probably the orientations will be represented internally as matrices, so a conversion step follows. The matrices are "lifted" to a sequence of neighboring quaternions, q_n, on the unit sphere. Each quaternion within the sequence will become the endpoint of two spherical Bézier curves. Between each quaternion pair, q_n and q_{n+1}, two additional points, a_n and b_{n+1}, are added to control motion through the joints. At this point, time becomes a parameter along the composite curve. As the frame number increments, the parameter enters and leaves successive curve pieces. Within each piece a local version of the parameter is adjusted to run from 0 to 1. Now the Bézier geometric construction comes into play, producing an interpolated quaternion, q_{n+u}, from q_n, a_n, b_{n+1}, q_{n+1}, and the local parameter, u. Finally the mint-fresh interpolated quaternion is transmuted into a matrix, to be used in rotating a list of object vectors for rendering.

5.2 Properties

A look at one special case is revealing. Suppose all the points to interpolate are spread along a single arc. This means they represent different amounts of rotation around a single axis, in which case quaternion multiplication commutes. Under these special conditions, the formula for the curve sections reduces to

$$q_{n+u} = q_n^{(1-u)^3}\ a_n^{3(1-u)^2u}\ b_{n+1}^{3(1-v)u^2}\ q_{n+1}^{u^3}$$

When this is compared to the standard Bézier polynomial, $p_n(1-u)^3 + a_n 3(1-u)^2 u + b_{n+1} 3(1-u)u^2 + q_{n+1}u^3$, it is apparent that addition and multiplication have become multiplication and exponentiation. Of course, when the points are not on one arc, commutativity fails, so the formula looks much messier.

In the interesting restricted case when the points are spaced evenly and consecutively around an arc, the resulting animation behaves exactly as we would hope: we get smooth,. constant speed rotation around the appropriate axis. Notice that we can choose *any* axis for this rotation. This is clearly preferable to interpolation with Euler's angles, where the coordinate axes are special. A more subtle property of *all* quaternion interpolation is that the motion is independent of coordinate axes. So, for example, if we design a move, then rotate the coordinate system arbitrarily, the geometry of the motion will not change. Euler interpolants, unfortunately, will do wildly different things.

5.3 Applicability

Rotations in space are significantly more complicated than rotations in a plane. It is easy to deal with the latter, since only one parameter is involved. Quaternions are out of place in a plane. Joint control in robotics simulations has its own highly specialized body of techniques; and though quaternions have shown up in the literature, they seem less useful in that context. [Brady][Taylor] However, B.K.P. Horn has used a tessellation of the quaternion unit sphere to identify the orientation of an object from its extended Gaussian image; a good reference is [Brou]. Non-rigid motion obviously needs to be handled specially. But for moving a camera eye-point, and for many kinds of object motion, quaternion interpolation has strong advantages.

5.4 Comparisons and complaints

Cost advantages are difficult to estimate. Converting a matrix to a quaternion requires only one square root and three divides plus some adds, at worst. Converting back requires 9 multiplies and 15 adds. While the conversions don't use trigonometric functions, the arc proportioning does. For comparison, angle interpolation requires several trigonometric functions as well as quite a few multiplies and adds to create each interpolated matrix. My experience is that the Bézier scheme is comfortably fast enough for design work, which is the only time speed has mattered. (If, for some application, more speed is essential, non-spherical quaternion splines will undoubtedly be faster than angle interpolation, while still free of axis bias and gimbal lock.)

These interpolants are not perfect, of course. Like all interpolants, they can develop kinks between the interpolated points. There are simple algorithms for adding new sequence points to ordinary splines without altering the original curve [Boehm]; they do not work for this interpolant. And if these curves can be shown to satisfy some variational principal, it will be by chance. It is useful to do this, because any solution to an integral equation like that for splines admits subdivision [Lane et al]; minimum curvature between end points implies minimum curvature between intermediate points as well. Along these lines, Gabriel and Kajiya, motivated by quaternions, have been developing a technique to find splines on arbitrary Reimannian manifolds by solving differential equations. [Gabriel & Kajiya]

6. Questions

Future research could answer some interesting practical questions. What are these spherical Bézier curves? Is there some abstract characterization of them? Or is there some related interpolant that is well-characterized? In light of the success of the geometric adaptation approach, it appears reasonable to apply the idea to B-splines, which also have a known geometric evaluation technique. [Gordon & Riesenfeld] How do spherical B-splines behave? Is it possible to add new points to a sequence for either kind of curve without disturbing it? How? Can B-splines be made to interpolate, not just approximate, with a simple adjustment of control points? Is there a way to construct a curve parameterized by arc length? This would be very useful. What is the best way to allow varying intervals between sequence points in parameter space? Abandoning the unit sphere, one could work with the four-dimensional Euclidean space of arbitrary quaternions. How do standard interpolation methods applied there behave when mapped back to matrices? Note that we now have little guidance in picking the inverse image for a matrix, and that cusp-free $\mathbf{R}^4$ paths do not always project to cusp-free S^3 paths.

However these questions are answered, quaternion spline interpolants already offer a well-behaved improvement over traditional techniques. They are simple to use, simple to implement, robust, efficient, consistent, and flexible. More research would make them even more so.

7. Acknowledgments

This work was begun for an animation system I designed and implemented at Singer-Link. Several people there deserve thanks, but I especially thank Glenn Davis, who befriended me with his good humor and mathematical efforts as I struggled through trying times.

I prefer not to invent the wheel if I can find the plans; so I pestered Don Venhaus, Brian Barsky, Tom Duff, Lance Williams, and Jim Blinn, whom I thank for their time, their comments, and their assurances that they had not seen this particular wheel roll past PDI, Berkeley, Lucasfilm, NYIT, or JPL.

Thanks to everyone at Pacific Data Images for the interest and encouragement that got me started.

The folks at Ridge Computer were generous above and beyond the call of customer support in letting me use their Imagen typesetting system to produce this paper.

Lastly, I thank Nori Hall for commenting on numerous drafts, and more.

References

1. BÉZIER, P.E., *Numerical Control — Mathematics and Applications,* John Wiley and Sons, London (1972).
2. BOEHM, WOLFGANG, "Inserting new knots into B-spline curves," *Computer-Aided Design* **12**(4) pp. 199-201 (July 1980).
3. BRADY, MICHAEL, "Trajectory Planning," in *Robot Motion: Planning and Control,* ed. Michael Brady, John M. Hollerbach, Timothy L. Hohnson, Tomas Lozano-Perez and Matthew T. Mason,The MIT Press (1982).
4. BROU, PHILIPPE, "Using the Gaussian Image to Find the Orientation of Objects," *The International Journal of Robotics Research* **3**(4) pp. 89-125 (Winter 1984).
5. CAYLEY, ARTHUR, "On certain results relating to quaternions," *Philosophical Magazine* **xxvi** pp. 141-145 (February 1845).
6. COURANT, R. AND HILBERT, D., *Methods of Mathematical Physics, Volume I,* Interscience Publishers, Inc., New York (1953).
7. DAHLQUIST, GERMUND AND BJÖRCK, ÅKE, *Numerical Methods,* Prentice-Hall, Inc., Englewood Cliffs, N.J. (1974). Translated by Ned Anderson.
8. EULER, LEONHARD, "Decouverte d'un nouveau principe de mécanique (1752)," pp. 81-108 in *Opera omnia, Ser. secunda, v. 5,* Orell Füsli Turici, Lausannae (1957).
9. EULER, LEONHARD, "Du mouvement de rotation des corps solides autour d'un axe variable (1758)," in *Opera omnia, Ser. secunda, v. 8,* Orell Füsli Turici, Lausannae ().
10. GABRIEL, STEVEN A. AND KAJIYA, JAMES T., "Spline Interpolation in Curved Manifolds," , (1985). Submitted
11. GARDNER, MARTIN, *New Mathematical Diversions from Scientific American,* Fireside, St. Louis, Missouri (1971). Chapter 2
12. GOLDSTEIN, HERBERT, *Classical Mechanics, second edition,* Addison-Wesley Publishing Company, Inc., Reading, Mass. (1980). Chapter 4 and Appendix B.
13. GORDON, WILLIAM J. AND RIESENFELD, RICHARD F., "Bernstein-Bézier methods for the computer-aided design of free-form curves and surfaces," *J. ACM* **21**(2) pp. 293-310 (April 1974).
14. GORDON, WILLIAM J. AND RIESENFELD, RICHARD F., "B-spline curves and surfaces," in *Computer Aided Geometric Design,* ed. Robert E. Barnhill and Richard F. Riesenfeld,Academic Press, New York (1974).
15. HAMILTON, SIR WILLIAM ROWAN, "On quaternions; or on a new system of imaginaries in algebra," *Philosophical Magazine* **xxv** pp. 10-13 (July 1844).
16. HERSTEIN, I.N., *Topics in Algebra, second edition,* John Wiley and Sons, Inc., New York (1975).
17. KANE, THOMAS R., LIKINS, PETER W. AND LEVINSON, DAVID A., *Spacecraft Dynamics,* McGraw-Hill, Inc. (1983).
18. LANE, JEFFREY M., CARPENTER, LOREN C., WHITTED, TURNER, AND BLINN, JAMES F., "Scan line methods for displaying parametrically defined surfaces," *Comm. ACM* **23**(1) pp. 23-34 (January 1980).
19. MACLANE, SAUNDERS AND BIRKHOFF, GARRETT, *Algebra, second edition,* Macmillan Publishing Co., Inc., New York (1979).
20. MISNER, CHARLES W., THORNE, KIP S., AND WHEELER, JOHN ARCHIBALD, *Gravitation,* W.H. Freeman and Company, San Francisco (1973). Chapter 41 — Spinors.
21. MITCHELL, E.E.L. AND ROGERS, A.E., "Quaternion Parameters in the Simulation of a Spinning Rigid Body," in *Simulation The Dynamic Modeling of Ideas and Systems with Computers,* ed. John McLeod, P.E., (1968).
22. NEWMAN, WILLIAM M. AND SPROULL, ROBERT F., *Principles of Interactive Computer Graphics, second edition,* McGraw-Hill, Inc., New York (1979). Chapter 21 — Curves and surfaces.
23. PICKERT, G. AND STEINER, H.-G., "Chapter 8 — Complex numbers and quaternions," in *Fundamentals of Mathematics, Volume I — Foundations of Mathematics: The Real Number System and Algebra,* ed. H. Behnke, F. Bachmann, K. Fladt, and W. Süss, (1983). Translated by S.H. Gould.
24. SCHMEIDLER, W. AND DREETZ, W., "Chapter 11 — Functional analysis," in *Fundamentals of Mathematics, Volume III — Analysis,* ed. H. Behnke, F. Bachmann, K. Fladt, and W. Süss,MIT Press, Cambridge, Mass. (1983). Translated by S.H. Gould.
25. SCHOENBERG, I.J., "Contributions to the problem of approximation of equidistant data by analytic functions," *Quart. Appl. Math.* **4** pp. 45-99 and 112-141 (1946).
26. SMITH, ALVY RAY, "Spline tutorial notes," Technical Memo No. 77, Computer Graphics Project, Lucasfilm Ltd. (May 1983).
27. SÜSS, W., GERICKE, H., AND BERGER, K.H., "Chapter 14 — Differential geometry of curves and surfaces," in *Fundamentals of Mathematics, Volume II — Geometry,* ed. H. Behnke, F. Bachmann, K. Fladt, and W. Süss,MIT Press (1983). Translated by S.H. Gould.
28. TAYLOR, RUSSELL H., "Planning and Execution of Straight Line Manipulator Trajectories," in *Robot Motion: Planning and Control,* ed. Michael Brady, John M. Hollerbach, Timothy L. Hohnson, Tomas Lozano-Perez and Matthew T. Mason,The MIT Press (1982).

Appendix I—Conversions

I.1 Quaternion to matrix

Using the restriction that $w^2+x^2+y^2+z^2=1$ for a quaternion $q = [w,(x,y,z)]$, the formula for the corresponding matrix is

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

If the quaternion does not have unit magnitude, an additional 4 multiplies and divides, 3 adds, and a square root will normalize it. (For the matrix conversion, the square root can be avoided in favor of divides if desirable.) Now we can obtain the operation count for creating the matrix. Most terms of the entries are a product of two factors, one of which is doubled. So we proceed as follows. First double x, y, and z, and form their products with w, x, y, and z. That will take 3 adds and 9 multiplies. Then form the sum for each of the 9 entries using 1 add each, plus an extra add for each of the 3 diagonal elements, for a total of 12 adds. Thus 9 multiplies and 15 adds suffice to convert a unit quaternion to a matrix.

I.2 Matrix to quaternion

An efficient way to determine quaternion components w, x, y, z from a matrix is to use linear combinations of the entries M_{mn}. Notice that the diagonal entries are formed from the squares of the quaternion components, while off-diagonal entries are the sum of a symmetric and a skew-symmetric part. Thus linear combinations of the diagonal entries will isolate squares of components; sums and differences of opposite off-diagonal entries will isolate products among x, y, and z and products with w. Using off-diagonals risks dividing by a component that may be zero, or within ϵ (the machine precision) of zero. However we can avoid that pitfall, and easily compute all components as follows.

```
$w^2 = 1/4\ (1 + M_{11} + M_{22} + M_{33})$
$w^2 > \epsilon$ ?
  TRUE
    $w = \sqrt{w^2}$
    $x = (M_{23} - M_{32})\ /\ 4w$
    $y = (M_{31} - M_{13})\ /\ 4w$
    $z = (M_{12} - M_{21})\ /\ 4w$
  FALSE
    $w = 0$
    $x^2 = -1/2\ (M_{22} + M_{33})$
    $x^2 > \epsilon$ ?
      TRUE
        $x = \sqrt{x^2}$
        $y = M_{12}\ /\ 2x$
        $z = M_{13}\ /\ 2x$
      FALSE
        $x = 0$
        $y^2 = 1/2\ (1 - M_{33})$
        $y^2 > \epsilon$ ?
          TRUE
            $y = \sqrt{y^2}$
            $z = M_{23}\ /\ 2y$
          FALSE
            $y = 0$
            $z = 1$
```

No more than one square root, three divides, and a few adds and binary scales are required for any conversion.

I.3 Euler angles to quaternion

There are twelve possible axis conventions for Euler angles. The one used here is ***roll***, ***pitch***, and ***yaw***, as used in aeronautics. A general rotation is obtained by first yawing around the **z** axis by an angle of ϕ, then pitching around the **y** axis by θ, and finally rolling around the **x** axis by ψ. Using the way quaternion components describe a rotation, we first obtain a quaternion for each simple rotation.

$$q_{roll} = [\cos\frac{\psi}{2},(\sin\frac{\psi}{2},0,0)]$$

$$q_{pitch} = [\cos\frac{\theta}{2},(0,\sin\frac{\theta}{2},0)]$$

$$q_{yaw} = [\cos\frac{\phi}{2},(0,0,\sin\frac{\phi}{2})]$$

Multiplying these together in the right order gives the desired quaternion $q = q_{yaw}\,q_{pitch}\,q_{roll}$, with components

$$w = \cos\frac{\psi}{2}\cos\frac{\theta}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\sin\frac{\theta}{2}\sin\frac{\phi}{2}$$

$$x = \sin\frac{\psi}{2}\cos\frac{\theta}{2}\cos\frac{\phi}{2} - \cos\frac{\psi}{2}\sin\frac{\theta}{2}\sin\frac{\phi}{2}$$

$$y = \cos\frac{\psi}{2}\sin\frac{\theta}{2}\cos\frac{\phi}{2} + \sin\frac{\psi}{2}\cos\frac{\theta}{2}\sin\frac{\phi}{2}$$

$$z = \cos\frac{\psi}{2}\cos\frac{\theta}{2}\sin\frac{\phi}{2} - \sin\frac{\psi}{2}\sin\frac{\theta}{2}\cos\frac{\phi}{2}$$

I.4 Euler angles to matrix

Combining the results of the previous two conversions gives

$$M = \begin{pmatrix} \cos\theta\cos\phi & \cos\theta\sin\phi & -\sin\theta \\ \sin\psi\sin\theta\cos\phi-\cos\psi\sin\phi & \sin\psi\sin\theta\sin\phi+\cos\psi\cos\theta & \cos\theta\sin\psi \\ \cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi & \cos\psi\sin\theta\sin\phi-\sin\psi\cos\phi & \cos\theta\cos\psi \end{pmatrix},$$

where ψ, θ, and ϕ are the angles of roll, pitch, and yaw, respectively.

I.5 Matrix to Euler angles

While converting a matrix to a unit quaternion only involves the sign ambiguity of square roots, converting to Euler angles involves inverse trigonometric functions, as we can only directly determine the sin's and cos's of the angles. Some convention, such as principle angles, must be adopted. However interpolation paths will vary greatly, depending on choice of angles. Setting that problem aside, here's a way to extract the sin's and cos's. Looking at the previous equation, $\sin\theta$ can be read off directly as $-M_{13}$. Use the trigonometric identity $\cos\theta = \pm\sqrt{1-\sin^2\theta}$ to compute $\cos\theta$ to within a sign, which is the best we can do. Assuming $\cos\theta$ is not zero, obtain the sin's and cos's of the other angles from

$$\begin{aligned}
\sin\theta &= -M_{13} \\
\cos\theta &= \sqrt{1-\sin^2\theta} \\
\sin\psi &= M_{23} \,/\, \cos\theta \\
\cos\psi &= M_{33} \,/\, \cos\theta \\
\sin\phi &= M_{12} \,/\, \cos\theta \\
\cos\phi &= M_{11} \,/\, \cos\theta
\end{aligned}$$

If $\cos\theta$ is zero, then we must avoid dividing by zero. It also becomes impossible to distinguish roll from yaw. Adopting the convention that the yaw angle ϕ is 0 allows

$$\begin{aligned}
\sin\psi &= -M_{32} \\
\cos\psi &= M_{22} \\
\sin\phi &= 0 \\
\cos\phi &= 1
\end{aligned}$$

From these values a two argument $\tan^{-1}$ will give angles between $-\pi$ and $+\pi$, or 0 and 2π, or some other conventional range; take your pick. (For a faster conversion, just compute, say, $\sin^{-1}$ and check the sign of the cosine term with respect to $\cos\theta$.) Because of the uncertainties of square roots, inverse trigonometric functions, and yaw-roll separation, matrix to Euler angle conversion is inherently *very* ill-defined.

I.6 Quaternion to Euler angles

Use the most straight-forward approach: convert the quaternion to a matrix, then the matrix to Euler angles. Of course it is unnecessary to compute matrix elements that are never used. This conversion is also unavoidably ill-defined, as quaternions contain no more information about angles than matrices do.

Quaternions and SLERP

Verena Elisabeth Kremer

University of Saarbrücken, Department for Computer Science
Seminar Character Animation
Supervisor: Michael Kipp
SS 2008

Abstract. Computing rotations is a common problem in both computer graphics and character animation. In his paper about animating rotations with quaternion curves, Ken Shoemake introduced an algorithm using Quaternions, SLERP, and Bézier Curves to solve this. The present paper discusses Shoemake's algorithm and the underlying concepts of quaternions, SLERP and Bézier Curves.

Key words: Character Animation, Quaternions, SLERP, Bézier Curves

1 Introduction

To smoothly simulate the motion of an animated character's joint (and the attached limb), one needs to compute the rotation and transition of the joint and also the way in between the starting and the ending point of the movement. The latter computation is called the in-between, or in-betweening a movement.

In the present paper, we will focus on an algorithm to compute rotations and the in-between presented by Ken Shoemake in his paper "Animating Rotation with Quaternion Curves" [1]. Shoemake suggests the use of quaternions to compute the rotation and spherical linear interpolation (SLERP) on parametrized Bézier Curves to compute the in-between. We will discuss all and their use as well as Shoemake's grand scheme in this paper.

Quaternions are hyper complex, four-dimensional numbers used to replace rotation matrices. Shoemake uses them due to the fact that a rotation can be smoothly computed by multiplying quaternions. We will have a more detailed look at this in section 2.

In section 3 we will discuss the computation of the in-between. Shoemake uses spherical linear interpolation (SLERP), a simple, straight forward formula for the interpolation between two points on a sphere. We will present some critique and discuss alternatives like normalized linear interpolation.

SLERP only yields good results for the interpolation between two points, i.e. the starting and ending points of a single rotation. To compute a smooth movement along several rotations and several corners, Shoemake used parametrized Bézier Curves which we will discuss in sections 4 and 5.

2 Geometry of Rotations

We know that movements from one point to another in three dimensional space are linear transitions of vectors, and we are able to decompose one complex transition into several simpler ones: a translation along a straight line and a rotation by an angle about an axis.

For further use we assume as given a time t_0, a vector $\overrightarrow{u} \in \mathbb{R}^3$, and a translation $f : \mathbb{R}^3 \to \mathbb{R}^3$. Our aim is to compute not only a vector $\overrightarrow{u}' \in \mathbb{R}^3$ at time t_1 and $\overrightarrow{u}' = f(\overrightarrow{u})$, but also corresponding vectors for time t, $t_0 \leq t \leq t_1$.

The vectors $\overrightarrow{u}$ and $\overrightarrow{u}'$ might represent for instance a camera or joint location at time t_0 and t_1 respectively. A particular time t would thus refer to a frame in a picture sequence. It is then not difficult to picture the sought-after vectors as the data needed to in-between frames t_0 and t_1: this is exactly what we are after.

Rotations of a vector $\overrightarrow{x}$ in a vector space are defined as linear transformations R that fulfill the following properties:

1. $\forall R(\overrightarrow{x}) \exists R^{-1} : R^{-1}(R(\overrightarrow{x})) = \overrightarrow{x}$
2. $R(\overrightarrow{x}) \cdot R(\overrightarrow{y}) = \overrightarrow{x} \cdot \overrightarrow{y}$
3. $R(\overrightarrow{x}) \times R(\overrightarrow{y}) = R(\overrightarrow{x} \times \overrightarrow{y})$

Rotations defined like this form with addition, scalar-, and vector product the space of rotations. The unit element is a rotation by $0°$. The inverse of a rotation by $\alpha°$ is the rotation by $(-\alpha)°$. We will discuss the space of rotations again later on.

A real square matrix whose transpose is its inverse and whose determinant is $+1$ fulfills the requirements listed above and is therefore a valid representation of a rotation. Matrices like this are also called rotation matrices. They are a very common way to represent rotations.

Rotation matrices conveniently correspond to geometric rotations about a fixed origin in an n-dimensional Euclidean space equipped with an Euclidean inner product (the matrix R being $\in \mathbb{R}^{n \times n}$). Thus multiplying the matrix with a n-dimensional real vector results in rotating the vector around an axis in this space. The Euclidean inner product of a real vector and a rotation matrix results in a vector of same size, shape, and handedness. In the three-dimensional space $\mathbb{R}^3$ we have

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix},\ R_y(\psi) = \begin{bmatrix} \cos\psi & 0 & \sin\psi \\ 0 & 1 & 0 \\ -\sin\psi & 0 & \cos\psi \end{bmatrix},\text{ and}$$

$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

For example, to perform a 90° - rotation of a vector $\overrightarrow{v} \in \mathbb{R}^3$ around the x-axis, we compute $Rot_x(\psi) \times \overrightarrow{v}$ with $\psi = 90°$.

3 Euler Angles

In order to precisely define the orientation of a rigid object in space Euler Angles can be used. Even though they may be used to compute rotations, they are originally designed to provide a clear and graphic description of the spatial orientation of a rigid object in space. Euler Angles are a set of angles between a rotated system – the one to be described – and a static system – the space the object is situated in. Euler angles are of great importance to physics, especially classical mechanics. In this context, the static system might be referred to as space coordinates while the moving system is called the body coordinates.

The fixed system consists of three axis, x, y, and z. It is associated with the Euclidean space. The rotated system consists of the axis X, Y, and Z. When no rotation has taken place, both systems are equal; $x = X$ and so on. As soon as the object is being moved, the XYZ-system rotates accordingly while the xyz system remains fixed.

The orientation of the object may now be described through three angles α, β and γ while

$$\alpha \overset{\text{def}}{=} \sphericalangle(\textit{x-axis},\ \textit{line of nodes}) \pmod{2\pi} \tag{1}$$

$$\beta \overset{\text{def}}{=} \sphericalangle(\textit{z-axis},\ \textit{Z-axis});\ 0 \le \beta \le \pi \tag{2}$$

$$\gamma \overset{\text{def}}{=} \sphericalangle(\textit{line of nodes},\ \textit{X-axis}) \pmod{2\pi} \tag{3}$$

There are several feasible definitions; this definition is used quite often and referred to in literature as the z-x-z convention.

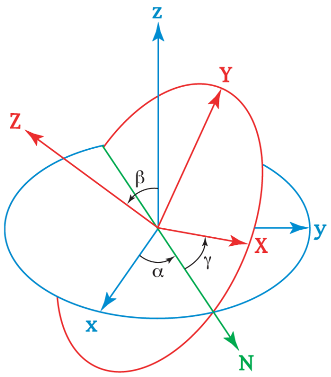


Fig. 1. The z-x-z convention; N denotes the line of nodes.

4 Quaternions

Shoemaker suggests one should use hyper complex four-dimensional numbers called quaternions to compute rotations and to perform SLERP with. Quaternions were invented by the Irish mathematician, philosopher, and astronomer Sir William Rowan Hamilton [5]. Quaternions form a group under multiplication called the Hamilton algebra [2].

The Hamilton algebra is defined as $\mathbb{H} = \{t + xi + yj + zk | t, x, y, z \in \mathbb{R}\}$ with the three imaginary units i, j, k:

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

For the sake of convenience the following notations will be used:

- a quaternion is written in **bold** text and might have an index: $\mathbf{q}_n \in \mathbb{H};\ n \in \mathbb{Q}$
- the three imaginary units are written as i, j, and k
- we write the imaginary part of a quaternion $\mathbf{q}$ as $\overrightarrow{q} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$
- a real number is called a scalar and is written as t_n; $n \in \mathbb{Q}$ is an index number

⇒ a quaternion $\mathbf{q}$ might therefore also be written as

$$\mathbf{q} = t + (i, j, k)\overrightarrow{q};\quad t \in \mathbb{R}$$

The multiplication table of the three imaginary units i, j, and k:
$ij = k \qquad jk = i \qquad ki = j$
$ik = -j \quad ji = -k \quad kj = -i$

Quaternion are multiplied by the so called Graham Product. It is defined as

$$\begin{aligned} \overrightarrow{q}_1 \overrightarrow{q}_2 &= -1(\overrightarrow{q}_1 \cdot \overrightarrow{q}_2) + \overrightarrow{q}_1 \times \overrightarrow{q}_2 \\ \mathbf{q}_1\mathbf{q}_2 &= t_1 t_2(-1)(\overrightarrow{q}_1 \cdot \overrightarrow{q}_2) + (i,j,k)(t_1\overrightarrow{q}_2 + t_2\overrightarrow{q}_1 + (\overrightarrow{q}_1 \times \overrightarrow{q}_2)) \end{aligned}$$

Note that quaternion multiplication is not commutative up to the case of $|\mathbf{q}| \in \{1, 0\}$. This can be a significant error-source, especially in code which computes several rotations at once. To avoid this, normalized quaternions can be used.

These formulas can be derived from vector multiplication using the standard definitions of inner and outer product as in vector space on $\mathbb{R}$ [4].
The identity quaternion number is 1 and it corresponds to the identity rotation in the space of rotations.

An inverse element can be constructed for every quaternion number:

$$\forall \mathbf{q} \in \mathbb{H} : \mathbf{q}^{-1} = t - (i,j,k)\overrightarrow{q}$$

Furthermore, the norm on $\mathbb{H}$ is given by $|\mathbf{q}| = \sqrt{\mathbf{q}q^{-1}}$.

5 Quaternions Used as Rotation Representation

Quaternions provide a straight forward way to represent, illustrate and compute rotations. This is due to the fact that quaternions can be used as a different and easier representation then rotation matrices; and a rotation can be smoothly computed by multiplying a quaternion with another. Since describing a single smooth rotation motion in three dimensional space is a principal task in computer animation, quaternions are of particular use to computer animation.

To illustrate this, we need to take a deeper look on how the space of rotation is constructed. Every rotation in a three-dimensional space is a rotation by some angle about some axis. The identity rotation is a rotation about any axis by the angle 0°. The set of possible rotations by any angle $\alpha \neq 0°$ can be described as a unit sphere $S(\alpha)$; and to describe three-dimensional rotations we need a hypersphere. A hypersphere is defined as a four-dimensional sphere with a three-dimensional surface and can be described by quaternions, since unit quaternions represent by themselves a unit hypersphere:

$$S^3 = \{\mathbf{q} \in \mathbb{H} : |\mathbf{q}| = 1\}$$

Computing the rotation of a quaternion is thus very easy: we only need to multiply it with the rotation matrix' hyper complex representation, i. e. with another quaternion [1]: a vector $\overrightarrow{v} \in \mathbb{R}^3$ can be rotated by a quaternion $\mathbf{q}$ by computation of $\mathbf{v} = (i, j, k)\overrightarrow{v}$ and

$$\overrightarrow{v}' = Rot(\overrightarrow{v}) = \mathbf{q}\mathbf{v}\mathbf{q}^{-1}$$

The proof of a unit quaternion qualifying as rotation can be found in the appendix.

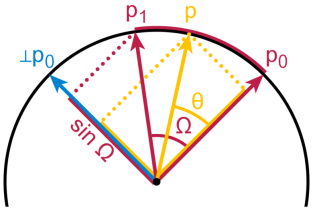
6 The In-Between – LERP, SLERP, and NLERP

Having computed a rotation successfully, whether using quaternions, rotation matrices, or Euler angles, one needs to think about the in-between. When a joint goes from one place to another, what does it look like? How can we compute the path this joint will take, and the points situated alongside?

The easiest way to interpolate between two points is the use of linear interpolation (LERP). LERP is a basic geometric formula: given the starting- and ending-points p_0 and p_1 and the interpolation parameter $t \in [0, 1]$, LERP(p_0, p_1, t) yields for each t a point along the straight line connecting them.

$$\text{LERP}(p_0, p_1; t) = (1 - t)p_0 + tp_1.$$

A straight line is however not useful to animations since a rotating joint is supposed to move along a smooth curve. For this, one can use spherical linear interpolation (SLERP). SLERP is LERP, performed on the surface of a unit sphere:



$$\text{SLERP}(p_0, p_1; t) = \frac{\sin{(1-t)\Omega}}{\sin \Omega} p_0 + \frac{\sin t\Omega}{\sin \Omega} p_1 \quad (4)$$

$$\text{SLERP}(\mathbf{q}_1, \mathbf{q}_2, t) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t \quad (5)$$

Using SLERP we get $p_t = \text{SLERP}(p_0, p_1; t)$, a point along the great circle arc on the surface of the unit sphere SLERP was performed upon.

NLERP finally stands for normalized (quaternion) SLERP:

$$\text{NLERP}(|\mathbf{q}_1|, |\mathbf{q}_2|, t) = |\mathbf{q}_1|(|\mathbf{q}_1|^{-1}|\mathbf{q}_2|)^t.$$

Since the quaternions are normalized NLERP is commutative, which SLERP is not [3]. It is therefor not as error-prone as SLERP is and it is also slightly faster. But it has some disadvantages as well: since the vectors are all of the same length, the computed points in between are not as evenly arranged. A joint following a NLERP computed curve is bound to move at the beginning and at the end faster then in the middle of the movement, while SLERP ensures constant velocity. Thus NLERP can be seen as the “quick ’n’ dirty” alternative; easy, but not always reliable enough.

7 Bézier Curves

If there are only two points between to interpolate, we simply calculate SLERP$(\mathbf{q}_1, \mathbf{q}_2, t)$ – interpreting t as a descrete time parameter (the number of the current frame). But what if we want to interpolate not two, but several points on a sphere, i.e. three ($\mathbf{q}_{n-1}$, $\mathbf{q}_n$, and $\mathbf{q}_{n+1}$)?

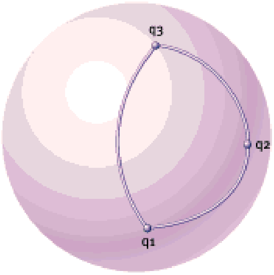


Fig. 2. Three points given[7]

We could interpolate first $\mathbf{q}_{n-1}$ and $\mathbf{q}_n$ and afterwards (separately) $\mathbf{q}_n$ and $\mathbf{q}_{n+1}$. We would get valid results, the algorithm would work. Unfortunately we would also remain with a non-differentiable curve. To avoid this, Shoemaker suggests the use of the parametrized Bézier curves.

Bézier curves are parametric and always differentiable curves named after Pierre Bézier, who used them to design automobile bodies[6]. Nevertheless not Bézier but Paul de Casteljau was the first to construct them. The curves are dependent on a number of parameters, three in a two-dimensional space and four in a three-dimensional environment.

We assume the three successive key quaternions $\mathbf{q}_{n-1}$, $\mathbf{q}_n$, and $\mathbf{q}_{n+1}$ as given. To build a Bézier curve with them, we have to

- compute two additional quaternion numbers, $\mathbf{a}_n$, and $\mathbf{b}_n$,

- draw a Bézier curve with the parameters $\mathbf{q}_n$, $\mathbf{a}_n$, $\mathbf{b}_n$, and $\mathbf{q}_{n+1}$,
- apply SLERP to the quaternions on this curve, computing the desired number of in-between points,
- and convert the data back into the original format.

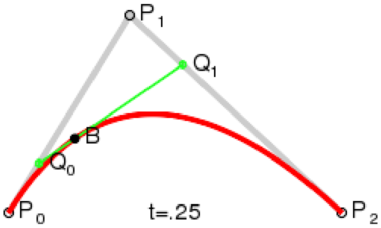


Fig. 3. An example for a two-dimensional Bézier curve [8].

To construct the quaternions $\mathbf{a}$ and $\mathbf{b}$ we approximate the derivative of a sampled function (with $\mathbf{q}_{n-1}$, $\mathbf{q}_n$, and $\mathbf{q}_{n+1}$ as samples) by averaging the central differences between $\mathbf{q}_{n-1}$, $\mathbf{q}_n$, and $\mathbf{q}_{n+1}$. We do so using the formula Shoemaker presents in [1]:

$$\begin{aligned} \mathrm{Biscet}(\mathbf{p},\mathbf{q}) &= \frac{\mathbf{p}+\mathbf{q}}{|\mathbf{p}+\mathbf{q}|} \\ \mathrm{Double}(\mathbf{p},\mathbf{q}) &= 2*(\mathbf{p}*\mathbf{q})*\mathbf{q}-\mathbf{p} \end{aligned}$$

$$\begin{aligned} \mathbf{a}_n &= \mathrm{Biscet}(\mathrm{Double}(\mathbf{q}_{n-1},\mathbf{q}_n),\mathbf{q}_{n+1}) \\ \mathbf{b}_n &= \mathrm{Double}(\mathbf{a}_n,\mathbf{q}_n) \end{aligned}$$

To construct the spherical Bézier curve with the quaternion parameters $\mathbf{q}_n$, $\mathbf{a}_n$, $\mathbf{b}_n$, and $\mathbf{q}_{n+1}$, we compute

$$\begin{array}{l} \mathbf{p}_{0,0} = \quad \mathbf{q}_n, \quad \mathbf{p}_{1,0} \quad = \quad \mathbf{a}_n, \\ \mathbf{p}_{2,0} = \mathbf{b}_{n+1}, \text{ and } \mathbf{p}_{3,0} = \mathbf{q}_{n+1} \end{array}, \quad \begin{aligned} \mathbf{p}_{0,1} &= \mathrm{SLERP}(\mathbf{p}_{0,0},\mathbf{p}_{1,0},t), \\ \mathbf{p}_{1,1} &= \mathrm{SLERP}(\mathbf{p}_{1,0},\mathbf{p}_{2,0},t), \\ \mathbf{p}_{2,1} &= \mathrm{SLERP}(\mathbf{p}_{2,0},\mathbf{p}_{3,0},t), \\ \mathbf{p}_{0,2} &= \mathrm{SLERP}(\mathbf{p}_{0,1},\mathbf{p}_{1,1},t), \\ \mathbf{p}_{1,2} &= \mathrm{SLERP}(\mathbf{p}_{1,1},\mathbf{p}_{2,1},t), \end{aligned}$$

and finally
$\mathbf{p}_{0,3} = \mathrm{SLERP}(\mathbf{p}_{0,2},\mathbf{p}_{1,2},t) =: \mathbf{q}_{n+t}$
while t goes from 0 to 1 in steps as big (or small) as we desire.[1] The smaller the steps, the more calculations are needed and the smoother the result will be.

8 Conclusion

Lets assume we want to perform a rotation on a joint described through vectors or Euler Angles in three-dimensional space; and we want to use Shoemake's suggestions. How does his algorithm looks like if everything is put together?

Shoemake's Grand Scheme

1. Since we want to work exclusively in $\mathbb{H}$, we need to convert the original data into a quaternion $\mathbf{q}_n$.
2. We design a unit quaternion $\mathbf{r}_n$ or convert a rotation matrix into $\mathbf{r}_n$.
3. To perform the rotation itself, we compute $\mathbf{q}_{n+1} = \mathbf{r}_n \mathbf{q}_n (\mathbf{r}_n)^{-1}$
4. We repeat for any other rotations $\mathbf{r}_{n+1}$, $\mathbf{r}_{n+2}$, etc.
5. If we stopped after the first rotation, we compute the in-between via $\mathrm{SLERP}(\mathbf{q}_n, \mathbf{q}_{n+1}, t_m) = \mathbf{q}_1(\mathbf{q}_1^{-1}\mathbf{q}_2)^t$ with $t \in [0,1]$, yielding the numbers $\mathbf{q}_n, \mathbf{q}_{n+t_1}, \mathbf{q}_{n+t_2}, \ldots, \mathbf{q}_{n+1}$ along a smooth curve.
6. If we performed several rotations, we construct a Bézier curve to compute SLERP upon (see section 7), yielding, again, the numbers $\mathbf{q}_n, \mathbf{q}_{n+t_1}, \mathbf{q}_{n+t_2}, \ldots, \mathbf{q}_{n+1}$ along a smooth curve, but also the numbers $\mathbf{q}_{n+1+t_1}, \mathbf{q}_{n+1+t_2}, \ldots \mathbf{q}_{n+2}, \ldots, \mathbf{q}_{n+3}, \ldots$
7. Afterwards we have a set of quaternion numbers along the path the joint takes while rotating. We only need to convert them back.

Since Shoemake introduced his idea about using quaternion SLERP on Bézier curves in 1985, the world of computation and programing has changed a lot. Thus it is even more remarkable that his algorithm is still considered important and up-to-date.

References

1. Ken Shoemake: Animating Rotation with Quaternion Curves. SIGGRAPH, Volume 19, Number 3 (1985)
2. Michael E. Mortenson: Mathematics for Computer Graphics Applications. Industrial Press, New York (1999)
3. Falk Reinhardt, Soeder: Atlas der Mathematik, Band 1. Deutscher Taschenbuch Verlag, Mnchen (2001)
4. Sir William Hamilton: On quaternions, or on a new system of imaginaries in algebra. Philosophical Magazine xxv, Dublin (1844)
5. Thomas Hankins: Sir William Rowan Hamilton. John Hopkins University Press, Baltimore (1980)
6. Pierre Bézier: Numerical Control – Mathematics and Applications. John Wiley and Sons, London (1972)
7. Image by Nick Bobick, wiki commons (2004)
8. Image by Phil Tregoning, wiki commons (2007)

9 Appendix: Unit Quaternions Represent Rotations

Assuming the computation of a quaternion rotation to be $R(\mathbf{x}) = \mathbf{r}\mathbf{x}\mathbf{r}^{-1}$, we have to show $\forall \mathbf{r}, |\mathbf{r}| = 1$ and $\forall \mathbf{x}, \mathbf{y}; \Re\mathbf{x} = \Re\mathbf{y} = 0$ that the following equations hold:

1. $\exists R^{-1} : R^{-1}(R(\mathbf{x})) = \mathbf{x}$
2. $R(\mathbf{x}) \cdot R(\mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$
3. $R(\mathbf{x}) \times R(\mathbf{y}) = R(\mathbf{x} \times \mathbf{y})$

.

To prove $\exists R^{-1} : R^{-1}(\mathbf{x})) = \mathbf{x}$, we can derive from $R^{-1} = \mathbf{r}^{-1}$:

$$\begin{aligned}
R^{-1}(R(\mathbf{x})) &= (\mathbf{r}^{-1})(R(\mathbf{x}))(\mathbf{r}^{-1})^{-1} && (6)\\
&= (\mathbf{r}^{-1})((\mathbf{r})(\mathbf{x})(\mathbf{r}^{-1}))(\mathbf{r}) && (7)\\
&= (\mathbf{r}^{-1}(\mathbf{r}))(\mathbf{x})((\mathbf{r}^{-1})(\mathbf{r})) && (8)\\
&= (\mathbf{x}) && (9)\\
& && (10)
\end{aligned}$$

Prove of $R(\mathbf{x}) \cdot R(\mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$:

$$\begin{aligned}
R(\mathbf{x}) \cdot R(\mathbf{y}) &= \frac{1}{2}(R(\mathbf{x})(R(\mathbf{y}))^{-1} + R(\mathbf{y})(R(\mathbf{x}))^{-1}) && (11)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x}\mathbf{r}^{-1})(\mathbf{r}\mathbf{y}\mathbf{r}^{-1})^{-1} + (\mathbf{r}\mathbf{y}\mathbf{r}^{-1})(\mathbf{r}\mathbf{x}\mathbf{r}^{-1})^{-1}) && (12)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x}\mathbf{r}^{-1})(\mathbf{r}\mathbf{y}^{-1}\mathbf{r}^{-1}) + (\mathbf{r}\mathbf{y}\mathbf{r}^{-1})(\mathbf{r}\mathbf{x}^{-1}\mathbf{r}^{-1})) && (13)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x})(\mathbf{r}^{-1}\mathbf{r})(\mathbf{y}^{-1}\mathbf{r}^{-1}) + (\mathbf{r}\mathbf{y})(\mathbf{r}^{-1}\mathbf{r})(\mathbf{x}^{-1}\mathbf{r}^{-1})) && (14)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x}\mathbf{y}^{-1}\mathbf{r}^{-1}) + (\mathbf{r}\mathbf{y}\mathbf{x}^{-1}\mathbf{r}^{-1})) && (15)\\
&= \mathbf{r}(\frac{\mathbf{x}\mathbf{y}^{-1} + \mathbf{y}\mathbf{x}^{-1}}{2})\mathbf{r}^{-1} && (16)\\
&= \mathbf{x} \cdot \mathbf{y}. && (17)\\
& && (18)
\end{aligned}$$

Prove of $R(\mathbf{x}) \times R(\mathbf{y}) = R(\mathbf{x} \times \mathbf{y})$:

$$\begin{aligned}
R(\mathbf{x}) \times R(\mathbf{y}) &= \frac{1}{2}(R(\mathbf{x})R(\mathbf{y}) - R(\mathbf{y})R(\mathbf{x})) && (19)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x}\mathbf{r}^{-1})(\mathbf{r}\mathbf{y}\mathbf{r}^{-1}) - (\mathbf{r}\mathbf{y}\mathbf{r}^{-1})(\mathbf{r}\mathbf{x}\mathbf{r}^{-1})) && (20)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x})(\mathbf{r}^{-1}\mathbf{r})(\mathbf{y}\mathbf{r}^{-1}) - (\mathbf{r}\mathbf{y})(\mathbf{r}^{-1}\mathbf{r})(\mathbf{x}\mathbf{r}^{-1})) && (21)\\
&= \frac{1}{2}((\mathbf{r}\mathbf{x})(\mathbf{y}\mathbf{r}^{-1}) - (\mathbf{r}\mathbf{y})(\mathbf{x}\mathbf{r}^{-1})) && (22)\\
&= \frac{1}{2}((\mathbf{r}(\mathbf{x}\mathbf{y})\mathbf{r}^{-1}) - (\mathbf{r}(\mathbf{y}\mathbf{x})\mathbf{r}^{-1})) && (23)\\
&= \mathbf{r}(\frac{\mathbf{x}\mathbf{y} - \mathbf{y}\mathbf{x}}{2})\mathbf{r}^{-1} && (24)\\
&= R(\mathbf{x} \times \mathbf{y}) && (25)\\
& && (26)
\end{aligned}$$