Enter the 3D Programming

第7回

2001.4

回れクォータニオン

久保裕一郎/宇治社中(http://www.cc.rim.or.jp/~devilman/)

3Dプログラミングにおいてよく見かけるクォータニオンとはいったいどのようなものでしょうか。今回はこのクォータニオンについて，その具体的な内容や効果，使い方について解説します。

はじめに

3Dプログラミングに興味を持ち始めると，必ずといっていいほど出くわすのがクォータニオン(quaternion)です。その名前から受ける印象や3Dに対する特効薬だという噂から，まるで秘儀のような扱いを受けることが多い話題です。でも，いざ調べてみるとあいまいな説明が多かったり，詳細で難解だったりと両極端で，これまた秘儀中の秘儀といった印象を受けがちです。

はたして本当にクォータニオンって難しいのでしょうか。たしかに厳密に議論するとなると理解できそうにないものですが，私たちの目的は3Dプログラミングで利用することなので，そこまでこだわる必要もなさそうです。だからといって，ブラックボックスとしてしか使えない程度の理解では，ちょっと物足りない気がします。

そこで今回は，この中間を狙うことにしましょう。

何に効くのか

まずは「何に効くのか？」という話から始めましょう。クォータニオンとは何ぞやという説明はあとにして，ここではクォータニオンを使うとこんなことができるらしいという情報を仕入れておきましょう。

ジンバルロック

ジンバルロック(Gimbal lock)という言葉を聞いたことはありませんか？　CGツールなどでアニメーションを作成している方ならすぐにピンとくるでしょう。これはあるやっかいな現象を意味する言葉です。きげんよく回転していた3Dオブジェクトが，あるとき予想していた回転を行わなくなってしまう，つまりロックしてしまうのです。

変なたとえですが，首を振る扇風機を思い浮かべてください。首を振るのがY軸回転，上下に傾けるのがX軸回転，そして羽根の回転がZ軸回転ですね。そこでおもむろにこの扇風機の首を真上に傾けてみてください。X軸回転90度で，風は真上に向かって吹いています。

まだ首振りスイッチはONになっています。でも，意味がなさそうに見えませんか？だってどれだけ首を振っても，風向きはずっと真上ですよね。ジンバルロックっていうのはこのことです。Y軸とZ軸が一致してしまい，1軸が死んでしまったのですね。

何が問題だったのでしょうか。これはそもそもXYZ軸それぞれの回転を別に考えていたからいけないのです。今までの回転方法，これをオイラー角表現(Euler angles representation)と呼びますが，これがジンバルロックの原因なのです。

気にしたことがなかったかもしれませんが，ジンバルロックはいろいろなところに悪影響を及ぼしてしまいます。そしてクォータニオンなら，このジンバルロックが起きないというのが，売り文句の1つなのです。

補間

売り文句の2つ目をご紹介しましょう。それは球面線形補間(spherical linear interpolation)ができるという利点です。これもあるたとえで説明してみましょう。

モスクワ経由とかアンカレッジ経由という言葉を聞いたことがあると思います。航空機の航路に関する言葉ですね。日本から飛び立った飛行機がアメリカに降り立つまで，どれぐらい航路が選択できるか考えてみてください。気流や補給の問題はおいたとしても，無数の航路が考えられます。極端な話，南へ向かって飛んで南極を通り越してから東に向かっても，西に向かってから北上しても，アメリカに到着することは可能です。

しかし，最短距離というのは一意に決まるはずです。2点をつなぐ最短ルートは球面上の直線で結べます。ここでまたオイラー角表現の弱点が現れてしまいました。オ

イラー角表現では，よく目にする東西南北が直交した世界地図上で北上して東に向かうようなルートでしか補間ができないのです。そしてクォータニオンなら，最短ルートでの補間が可能になります。

クォータニオンにはオイラー角表現の弱点を補う効能があるみたいですね。いよいよ説明に入りたいところですが，その前に少し準備をしておかないといけません。

複素数

その準備とは複素数(complex number)のことです。ご存じの方も多いでしょうが，おさらいもかねて進んでみましょう。複素数とはFig. 1のような形の数のことです。

複素数はaで表される実部とbiで表される虚部から構成されています。ここで出てくるiを虚数(imaginary number)といい，これは2乗して-1になるという頭の痛い存在です。どれだけ想像しても思い描けないという意味で虚ということなのでしょうか。それでも数学者はこの複素数を図示する方法を考えました。それが複素平面(ガウス平面)です。

複素平面

複素平面はFig. 2の図を見ながら説明します。まず直交する2軸を決めて水平の軸を実軸と呼び，右を正として実部の値を表現します。また，垂直の軸を虚軸として上方向正で虚部の値を表現します。すると，平面上に2次元グラフとして複素数の位置を図示することができます。

なんか数学の授業みたいでつまらないかもしれませんが，この複素平面表示からはいくつかの重要な発見があります。まずは絶対値(absolute)という概念が生まれます。複素数の大きさといわれても理解しにくいですが，絶対値(長さ)といわれればもうベクトルでおなじみですよね。そうなんです，複素数の演算の多くは2次ベクトルとして取り扱うことが可能になるのです。

Fig. 3は，1やiに注目して図示したものです。実数1に虚数iを掛け算することを考えてみてください。答えは迷わずiです。**さらにiにiを掛けると-1に，-1掛けるiは-iになります。どんどんiを掛け続けただけです。出てきた結果を複素平面上で追いかけると……なんと半時計回りに回転しています。**ここでいう回転がすなわち3Dオブジェクトの回転ではありませんが，虚数の不思議さという意味で紹介しました。

クォータニオン

日本語では四元数と呼ばれるクォータニオンですが，そもそもこれは何なんだということを始めに説明しておきましょう。複素数の1段階上の概念というほうが理解しやすいでしょうが，ここでは少し乱暴に割り切った考え方をします。複素数が実数世界に虚数iという概念を導入して構築された数学の1分野だとしたとき，同様にi，j，kというルールを実数に導入して生まれた世界のことをクォータニオンだと考えましょう。

演算も性質も実数やベクトルとは異なり，ときには複素数とも異なる，新たなルールによる世界を学んでいきます。もちろん，努力の報酬として最初にあげた3Dでの効能を期待して話を進めていきましょう。

ルール

まずは定義から見てみます。クォータニオンとはFig. 4-(a)のような形をしています。ここでqがクォータニオンで，w，x，y，zはそれぞれ実数です。複素数と同じようなiだけでなくjやkも存在していますが，何となくやろうとしていることはわかるはずです。先ほどの複素数が実部と虚部とい

Fig. 1 複素数の式

$$z=a+bi$$
$$i^2=-1$$

Fig. 2 複素平面

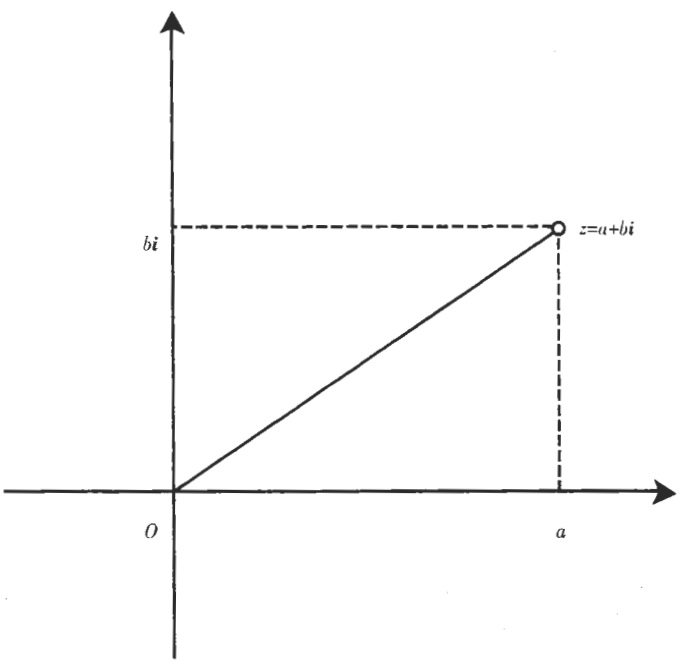


Fig. 3 1次元に注目した複素平面

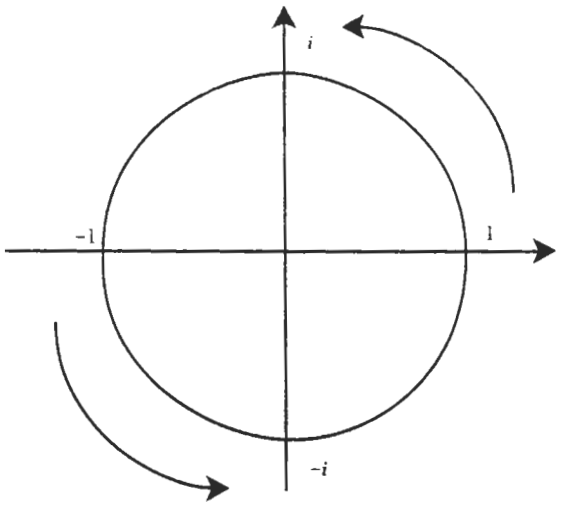


Fig. 4 クォータニオンのルール
(a)数式での表現

$$q=w+xi+yj+zk$$

(b)i, j, kのルール

$$ii=jj=kk=ijk=-1$$
$$ij=-ji=k$$
$$jk=-kj=i$$
$$ki=-ik=j$$

う2要素構成だったのに対して，クォータニオンは4要素構成というわけです。だから四元なんですね。

さて，気になるi，j，kですが，やはりちょっと理解しにくいルールになっています。まずはFig. 4-(b)を見ていただきましょう。同じものを掛け合わせると-1になるというのはいいとしても，それ以外は複雑になっています。違う種類を掛け合わせると残りの記号になったり，逆に掛けると符合が反転したりと何だか外積のにおいがしてきます。これをきちんと理解するには誌面が足りないので，その説明は割愛させていただきます。ここでは，こんなルールで話が進むのだということを確認しておいてください。

実はクォータニオンの初期ルールはこれだけです。この初期ルールに従っていろいろと演算方法や特性を考えていくことになります。

幸い私たちにはベクトルに関しての知識が十分にあるので，それを最大限に生かして進むことにしましょう。

加減乗算

演算方法として最初に行うのは加減算ですが，まさに先ほどいったベクトルでおなじみの演算方法です。

Fig. 5-(a)のような2つのクォータニオンがあるとき，それらを使って加減算するとFig. 5-(b)となります。各実数部分を要素ごとに足したり引いたりするだけでよいので，違和感はありません。

気が楽になったところで乗算に進みましょう。まずはスカラー倍から始めてみます(Fig. 5-(c))。これもベクトルと同じような考え方ですね。あれ？　それほど難しくないですね。では，クォータニオン同士の乗算ならどうかというと，ベクトルのように複数の方法があるわけでもなく素直に開いてやればいいだけなのです(Fig. 5-(d))。

理解しやすい手順とはいえあまりにも汚いので，ここでi，j，kによるルールを用いて整理すると，Fig. 5-(e)のようにほんの少しきれいな形に整理できます。初めてクォータニオンらしい手順が登場しました。それでもまだ覚えにくい形ですよね。

ベクトルを用いた表記

そこで，本物のベクトルを用いた表現を行ってみましょう。i，j，kそれぞれに掛かる係数を要素とした3次ベクトルvを用いて表記する方法です(Fig. 6-(a))。裸の実数であったwを残してベクトルに畳み込んだ感じです。このときのwの部分をスカラー部，vの部分をベクトル部と呼ぶこともあります。これを用いると前述の加減算や乗算はFig. 6-(b)のように驚くほどすっきりと表現できます。

とくに乗算の場合，やっぱり出てきた内積や外積がうまく利用されています……といいたいところですが，事実はその逆です。実数要素wが0の場合のクォータニオン「純四元数(pure quaternion)」を考えて，その乗算を行ってみてください。スカラー部は内積(scalar product)，ベクトル部は外積(vector product)となります。実はこのクォータニオンという概念，私たちが勉強してきた3Dベクトル空間という概念よりも前に成立していたのです。クォータニオンの世界に現れる3次ベクトルや内積外積を便利だからという理由で拝借して3D表現に利用したというのが本当のところのようです。

ここで注意しておくべき点として乗算の交換法則が成り立たないということがあります。q_1q_2とq_2q_1は必ずしも一致しないということです。外積が登場している時点で予想できたことですが，一応確認しておきましょう。

逆クォータニオン

さっそく残りの除算……といいたいところですが，交換法則が成り立たないときたら普通の除算はできそうにありません。行列の除算を逆行列の乗算で行っていたよう

Fig. 5 クォータニオンの加減乗算

(a)例となる数式

$q_1=w_1+x_1\boldsymbol{i}+y_1\boldsymbol{j}+z_1\boldsymbol{k}$

$q_2=w_2+x_2\boldsymbol{i}+y_2\boldsymbol{j}+z_2\boldsymbol{k}$

(b)加減算

$q_1+q_2=(w_1+w_2)+(x_1+x_2)\boldsymbol{i}+(y_1+y_2)\boldsymbol{j}+(z_1+z_2)\boldsymbol{k}$

$q_1-q_2=(w_1-w_2)+(x_1-x_2)\boldsymbol{i}+(y_1-y_2)\boldsymbol{j}+(z_1-z_2)\boldsymbol{k}$

(c)スカラー倍

$sq=sw+sx\boldsymbol{i}+sy\boldsymbol{j}+sz\boldsymbol{k}$

(d)乗算

$q_1q_2=(w_1+x_1\boldsymbol{i}+y_1\boldsymbol{j}+z_1\boldsymbol{k})(w_2+x_2\boldsymbol{i}+y_2\boldsymbol{j}+z_2\boldsymbol{k})$

$=w_1w_2+w_1x_2\boldsymbol{i}+w_1y_2\boldsymbol{j}+w_1z_2\boldsymbol{k}$

$+x_1\boldsymbol{i}w_2+x_1\boldsymbol{i}x_2\boldsymbol{i}+x_1\boldsymbol{i}y_2\boldsymbol{j}+x_1\boldsymbol{i}z_2\boldsymbol{k}$

$+y_1\boldsymbol{j}w_2+y_1\boldsymbol{j}x_2\boldsymbol{i}+y_1\boldsymbol{j}y_2\boldsymbol{j}+y_1\boldsymbol{j}z_2\boldsymbol{k}$

$+z_1\boldsymbol{k}w_2+z_1\boldsymbol{k}x_2\boldsymbol{i}+z_1\boldsymbol{k}y_2\boldsymbol{j}+z_1\boldsymbol{k}z_2\boldsymbol{k}$

(e)(d)を$\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$を用いて整理

$q_1q_2=(w_1w_2-x_1x_2-y_1y_2-z_1z_2)$

$+(w_1x_2+w_2x_1+y_1z_2-z_1y_2)\boldsymbol{i}$

$+(w_1y_2+w_2y_1+z_1x_2-x_1z_2)\boldsymbol{j}$

$+(w_1z_2+w_2z_1+x_1y_2-y_1x_2)\boldsymbol{k}$

Fig. 6 ベクトルを用いた表記方法

(a)3次ベクトル$\boldsymbol{v}$を用いた表記

$\boldsymbol{v}=<x, y, z>$

$q=(w, \boldsymbol{v})$

(b)Fig. 5の加減算に$\boldsymbol{v}$を用いた例

$q_1+q_2=(w_1+w_2, \boldsymbol{v}_1+\boldsymbol{v}_2)$

$q_1-q_2=(w_1-w_2, \boldsymbol{v}_1-\boldsymbol{v}_2)$

$q_1q_2=(w_1w_2-\boldsymbol{v}_1\cdot\boldsymbol{v}_2, w_1\boldsymbol{v}_2+w_2\boldsymbol{v}_1+\boldsymbol{v}_1\times\boldsymbol{v}_2)$

に，クォータニオンの除算も逆クォータニオンの乗算で行います。

逆クォータニオンを求めるために，いくつかの概念の説明を先にしておきましょう。まずは，絶対値です。これは複素数のときに触れたようにベクトルと同じように求めることができます(Fig. 7-(a))。直感どおりですね。もちろん答えはスカラー(実数)になります。いわば長さを求めることができたので，正規化ベクトルのような正規化クォータニオン(unit quaternion)というものも考えられます。スカラー倍が可能であるなら，スカラー除算も当然可能になるので，Fig. 7-(b)のように長さが1のクォータニオンを作ることができます。この長さ

Fig. 7 逆クォータニオン

(a)絶対値

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

(b)正規化クォータニオン

$normalize(q)=q/|q|$

(c)共役

$\bar{q}=(w, -\boldsymbol{v})$

(d)クォータニオンと共役を掛け合わせる

$q\bar{q}=(w, \boldsymbol{v})(w, -\boldsymbol{v})$

$=(w^2+\boldsymbol{v}\cdot\boldsymbol{v}, -w\boldsymbol{v}+w\boldsymbol{v}+\boldsymbol{v}\times(-\boldsymbol{v}))$

$=w^2+\boldsymbol{v}\cdot\boldsymbol{v}$

$=|q|^2$

(e)同じもので割る

$q\bar{q}/|q|^2=1$

(f)逆クォータニオンの数式

$q^{-1}=\bar{q}/|q|^2$

が1のクォータニオン，あとで回転のことを考える際に非常に重要になります。

また，共役(conjugate)というものも知っておきましょう。複素数にも存在しますが，共役とはスカラー部分以外にマイナスを掛けたクォータニオンのことを指します(Fig. 7-(c))。名前まで与えられているだけあって，この共役にはいろいろとおもしろい性質があります。たとえばクォータニオンとその共役を足せばスカラー部分が残るというのもそうですが，もっと利用価値の高いものとしてクォータニオンとその共役を掛け合わせてみましょう(Fig. 7-(d))。これも答えがスカラーになります。それだけではなく長さの2乗なんていうできすぎた答えが手に入りました。

そこで，ちょっとひねってFig. 7-(e)のような式を考えてみましょう。同じもので割っているので，答えは1になるのがあたりまえですが，この式はqに何かを乗算すると1になるということも表しています。つまり，ここで掛けているものが求めている逆クォータニオンです。Fig. 7-(f)のように，逆クォータニオンは共役を絶対値の2乗で割ったものになります。また，このクォータニオンが正規化クォータニオンだった場合，逆クォータニオンは共役そのものになります。

逆クォータニオンが手に入って，除算が行えるようになりましたので，不思議な$\boldsymbol{i}$, $\boldsymbol{j}$, $\boldsymbol{k}$から生まれた演算の仕組みはここまでになります。それにしても思っていたより難しいところはありませんでしたよね。内積や外積のほうがよっぽど理解しにくい気がします。

回転との関係

それではいよいよ3Dプログラミングの話題，回転の説明に進みます。クォータニオンが回転をどう表すのか，またどう取り扱うのかを見ていきます。

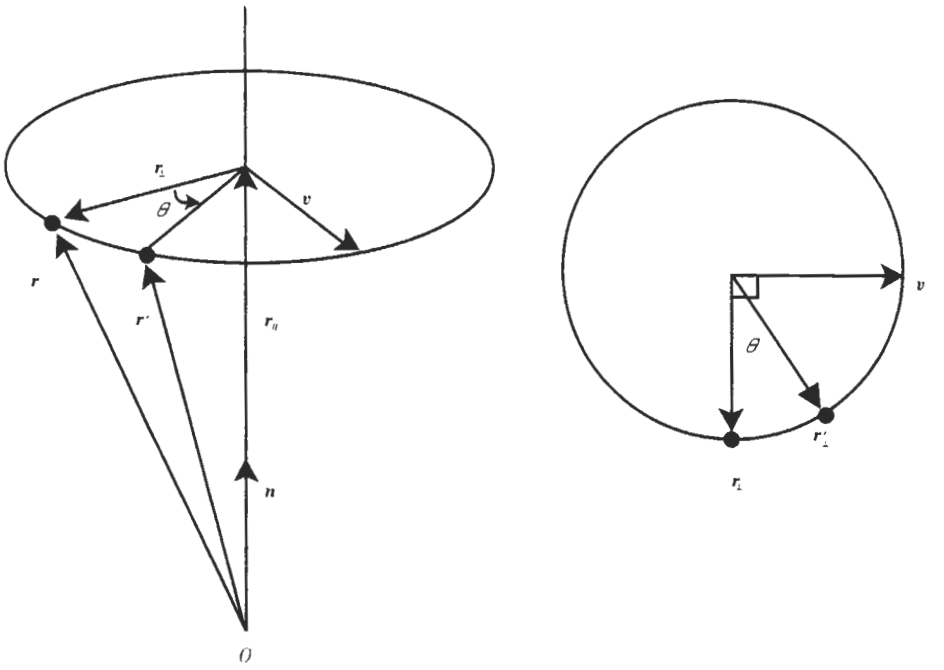
回転軸表現

いざクォータニオンでの回転に進む……その前に，回転軸表現(Axis-angle representation)の説明をしておきましょう。問題の多かったオイラー角表現では，*XYZ*各軸における回転角度によってオブジェクトの姿勢を表現していましたが，回転軸表現は回転軸となるベクトルと，回転角度によって姿勢の表現を行います。*XYZ*軸にとらわれない，自由な軸を中心に回転させることから，自由軸回転あるいは任意軸回転とも呼ばれます。

ここでFig. 8の図を見ながら考えていきましょう。回転軸となる正規化ベクトルを$\boldsymbol{n}$として角度θの回転を行うとします。回転の対象となる点を指すベクトルを$\boldsymbol{r}$として回転後のベクトルを$\boldsymbol{r}'$とします。

ここで話を簡単にするために，ベクトル$\boldsymbol{r}$を2つのベクトルに分解することを考えます。回転軸$\boldsymbol{n}$に沿った平行成分ベクトル$\boldsymbol{r}_{/\!/}$と，$\boldsymbol{n}$に対して垂直に回転面を構成する垂直成分$\boldsymbol{r}_{\perp}$に分割します。ベクトル成分の分解といえば内積の登場です。Fig. 9-(a)のように分割することによって，ベクトルの回

Fig. 8 回転軸



転を回転面のみでの計算に変換できます。Fig. 9-(a)から回転後も成分$r_{//}$に変化がないことは明らかです。Fig. 8の右側の図が回転面を上から見た図になります。

もう1つ補助になるベクトルを定義しておきましょう。それがvです。このベクトルは回転平面上にあり軸nに直行していて，$r_\perp$に対しても直行していることとします。このような2軸に直行しているベクトルといえば外積の登場です(Fig. 9-(b))。もともと$r_\perp$に関しては分解成分であり，nは正規化ベクトルですから，この補助ベクトルvはnとrとの外積で求めることができます。

これでもう話は回転平面上での説明になります。回転先ベクトルの垂直成分$r'_\perp$は$r_\perp$とvを利用してFig. 9-(c)のように表すことができ，回転により変化しない$r_{//}$と合成してやると，回転後のベクトルr'を作ることができます。これがFig. 9-(d)です。ちょっとわかりにくい式のような気がするかもしれませんが，このように回転軸nと回転角度θによってベクトル回転を表現できることがわかりました。実はこの回転軸表現であれば，問題であったジンバルロックは回避できるのです。「なんだ，じゃあそれでいいや」といわずにクォータニオンでの回転も見ていってください。

クォータニオンでの回転

さっそく始めましょう。まず，回転されるベクトルrをクォータニオンpで表すことにします(Fig. 10-(a))。純虚数のベクトル部によってベクトルを表現するわけですね。それとまだ正体は明かせませんが，ある正規化クォータニオンqを用意することにします。

これらを使って，Fig. 10-(b)のような変わった形の数式について考えてみましょう。この数式ではある正規化クォータニオンに純虚数を掛けて，それに最初の正規化クォータニオンの逆クォータニオンを掛けています。いったい何を行っているのか想像がつきません。そこでこれを展開して調査してみます。これには正規化クォータニオンの逆クォータニオンは，共役と等しくなるということが利用できます(Fig. 10-(c))。少し省略しましたが，最後の式に持ってくるまでに内積や外積の特性をうまく利用しています。時間のある方は実際に計算してみるといいでしょう。

何だかいわくありげな形まで持ってきたところ，正規化クォータニオンだとしか教えられていなかったqが，実はFig. 10-(d)のような形をしていたとします。ベクトルnが正規化ベクトルだという前提で代入を行ってみると，Fig. 10-(e)というふうに数式を変化させることができます。このよう

Fig. 9 ベクトルを2つに分解して回転を考える

(a)ベクトル成分の分解

$$r_{//}=(n\cdot r)n$$
$$r_\perp=r-(n\cdot r)n$$

(b)2軸に直行しているベクトル

$$v=n\times r_\perp=n\times r$$

(c)回転先ベクトルの垂直成分

$$r'_\perp=(\cos\theta)r_\perp+(\sin\theta)v$$

(d)回転後のベクトル

$$\begin{aligned}r'&=r_{//}+r'_\perp\\&=r_{//}+(\cos\theta)r_\perp+(\sin\theta)v\\&=(n\cdot r)n+(\cos\theta)(r-(n\cdot r)n)+(\sin\theta)n\times r\\&=(\cos\theta)r+(1-\cos\theta)n(n\cdot r)+(\sin\theta)n\times r\end{aligned}$$

Fig. 10 クォータニオンでの回転

(a)回転されるベクトルrをクォータニオンpで表す

$$p=(0, r)$$

(b)正規化クォータニオンqを用いて考える

$$p'=qpq^{-1}$$

(c)(b)を展開

$$\begin{aligned}qpq^{-1}&=(w, v)(0, r)(w, -v)\\&=(-v\cdot r, wr+v\times r)(w, -v)\\&=(0, (w^2+v\cdot v)r+2w(v\times r))\\&=(0, (w^2-v\cdot v)r+2v(v\cdot r)+2w(v\times r))\end{aligned}$$

(d)正規化クォータニオンqの内容

$$q=(\cos\phi, \sin\phi n)$$

(e)ベクトルnが正規化ベクトルだという前提で(d)を代入したもの

$$\begin{aligned}p'&=qpq^{-1}\\&=(0, (w^2-v\cdot v)r+2v(v\cdot r)+2w(v\times r))\\&=(0, (\cos^2\phi-\sin^2\phi)r+2\sin^2\phi n(n\cdot r)+2\cos\phi\sin\phi(n\times r))\\&=(0, \cos 2\phi r+(1-\cos 2\phi)n(n\cdot r)+\sin 2\phi(n\times r))\end{aligned}$$

(f)クォータニオンによる回転

$$q=(\cos(\theta/2), \sin(\theta/2)n)$$

Fig. 11 行列によるクォータニオンの表記

(a)クォータニオンの乗算を行列として表記

$$q_1q_2=\begin{vmatrix} w_1x_2-z_1y_2+y_1z_2+x_1w_2\\ z_1x_2+w_1y_2-x_1z_2+y_1w_2\\ -y_1x_2+x_1y_2+w_1z_2+z_1w_2\\ -x_1x_2-y_1y_2-z_1z_2+w_1w_2\end{vmatrix}=L(q_1)\begin{vmatrix}x_2\\y_2\\z_2\\w_2\end{vmatrix}$$

(b)$L(q)$の内容

$$L(q)=\begin{vmatrix} w & -z & y & x\\ z & w & -x & y\\ -y & x & w & z\\ -x & -y & -z & w\end{vmatrix}$$

(c)右から掛けることと同じ働きをする行列R

$$q_1q_2=R(q_2)q_1=\begin{vmatrix} w_2 & z_2 & -y_2 & x_2\\ -z_2 & w_2 & x_2 & y_2\\ y_2 & -x_2 & w_2 & z_2\\ -x_2 & -y_2 & -z_2 & w_2\end{vmatrix}\begin{vmatrix}x_1\\y_1\\z_1\\w_1\end{vmatrix}$$

(d)回転行列

$$qpq^{-1}=qp\bar{q}=L(q)R(\bar{q})p$$

$$=\begin{vmatrix} w^2+x^2-y^2-z^2 & 2(xy-wz) & 2(xz+wy) & 0\\ 2(xy+wz) & w^2-x^2+y^2-z^2 & 2(yz-wx) & 0\\ 2(xz-wy) & 2(yz+wx) & w^2-x^2-y^2+z^2 & 0\\ 0 & 0 & 0 & 1\end{vmatrix}\begin{vmatrix}x_v\\y_v\\z_v\\0\end{vmatrix}$$

に変化させると，任意軸回転を行うための数式とほとんど同じものが手に入ることが，ここからわかります。つまり，クォータニオンでも任意軸回転ができるのです。違うところは，結果が純クォータニオンであるということと，回転角度が2ϕになっていることだけです。

回転についてまとめてみましょう。任意の正規化ベクトルnを軸として回転角度θぶんの回転をベクトルrに適用するには，rを純クォータニオンpとして取り扱い，Fig. 10-(f)のようなクォータニオンqを用いて，qpq^{-1}の演算を行うということになります。

これがクォータニオンによる回転の正体だったのです。

行列での表記

クォータニオンによる回転を見てきて，どのような感想を抱かれたでしょうか。たぶんqpq^{-1}という演算に違和感があったのではないかと思います。こんな不自然な手順が加わるのなら，クォータニオンの採用は見送ろうかとさえ思ってしまいます。

そこで登場するのが行列による表記です。ある行列を用いるとベクトルに行列を掛けるだけでクォータニオンによる回転を行うことができるのです。

回転行列

まずは行列によって，クォータニオン同士の乗算を表現できないか考えてみます。クォータニオンの4つの要素をI軸，J軸，K軸，実軸の順番に並べたベクトルとして行列を作ってみます。Fig. 11-(a)の式を見てください。左側が乗算の計算結果を4要素のベクトルとして表したものです。並んでいる順序に注意しながら以前の乗算の式と同じものであることを確認してください。右側の$L(q_1)$に相当する行列を見つけ出すことによって，行列によるクォータニオン同士の乗算を表そうというわけです。

見つかりやすいように並べておきましたから，$L(q)$はすぐに求めることができますね。Fig. 11-(b)のようになります。この行列を作用させると，クォータニオンを左から掛けた場合と同じ結果を得ることができます。同様の考え方で，今度は右から掛けることと同じ働きをする行列$R(q)$を求めてみます(Fig. 11-(c))。

このように左から掛けるような行列と右から掛けるような行列の2つを用いて回転行列を作成できるのです。回転用のクォータニオンqは正規化クォータニオン，回転の対象であるpは純クォータニオンであることに留意しながら，Fig. 11-(d)という行列が手に入りました。長い道のりをたどってきましたが，どうやら一応の収穫があったみたいです。この4×4行列はまさに回転行列として使用することができます。回転したい軸と，その角度からx, y, z, wを算出し，このような行列を作成すると，ようやくそれでジンバルロックを起こさない回転方法が手に入るのです。

1つ注意を促しておきましょう。実軸を行列の4行目に持ってくることによってオイラー角での回転行列と同じような表現が可能になったのですが，同次座標系でいうところのwとはまったく意味が違いま

その証拠に，pの第4要素は0になっていますよね。とはいえ，この形の行列であれば，回転をつかさどる3×3部分でしか影響がありませんので，今までの行列演算にそのまま組み込んで使用することができます。

回転の合成

回転の説明のしめくくりとして，クォータニオンの回転は合成できるという話をしておきましょう。最初にq_1だけ回転して，続けてq_2回転を行う場合を考えます。するとFig. 12-(a)のような式で表すことができます。

2度回転の手順を行うことと，q_2q_1というクォータニオンで1度に回転を行うことはまったく同じになります。これを用いて，さらにクォータニオンの導入を楽にするいい方法があります。

Fig. 12-(b)のようなq_{tot}を作成して，それを回転行列に変換させれば，今までのオイラー角での3Dプログラミングへひそかにクォータニオンを潜り込ませることができます。*XYZ*軸回転で指定をしつつ，ジンバルロックも回避することが可能になり，インタフェイスはまったく変更することなしに扱うことができます。ただし，回転の順番には注意が必要です。

Fig. 12 回転の合成

(a)最初にq_1だけ回転して続けてq_2回転を行う場合

$$q_2(q_1pq_1^{-1})q_2^{-1}=(q_2q_1)p(q_1^{-1}q_2^{-1})=(q_2q_1)p(q_2q_1)^{-1}$$

(b)クォータニオンを回転行列に変換させる数式

$$q_1=(\cos\frac{\phi_1}{2}, <\sin\frac{\phi_1}{2}, 0, 0>)$$

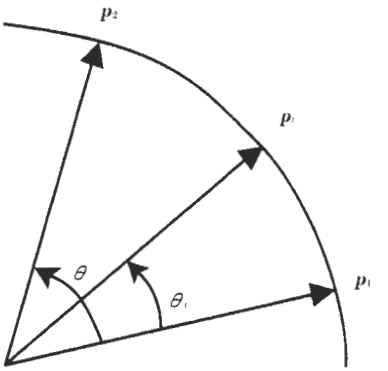
$$q_2=(\cos\frac{\phi_2}{2}, <0, \sin\frac{\phi_2}{2}, 0>)$$

$$q_3=(\cos\frac{\phi_3}{2}, <0, 0, \sin\frac{\phi_3}{2}>)$$

$$q_{tot}=q_3q_2q_1$$

Fig. 13 回転軸

(a)グラフ



(b)中間ベクトルp_tの数式

$$p_t=p_1\frac{\sin(\theta(1-t))}{\sin\theta}+p_2\frac{\sin(\theta t)}{\sin\theta}$$

補間の仕組み

クォータニオンによって任意軸による回転が行えるようになりました。最後の話題は補間に関する説明です。感覚的な説明を先にしておくと，ある回転を表現するクォータニオンq_1と別の回転q_2との間に，新たに中間のクォータニオンを作成することで補間を実現しています。また，そのようなクォータニオンを生成できること自体に意味がある場合もあります。

球面線形補間の説明を2次元で図示しながら行ってみます。Fig. 13-(a)の図のように，2次元ベクトルp_1とp_2が存在していて，それぞれ正規化された(長さが1の)ベクトルであるとします。tは0から1の任意の数であるとします。このとき中間ベクトルp_tは，Fig. 13-(b)という数式で表すことができます。中間ベクトルp_tはtが0に近いほどp_1寄りに位置し，tが1に近づくにつれ，p_2寄りに存在するという式です。この式の導き方に関しては，常微分や角速度などが登場してくるので，ここでは数式の紹介のみにとどめます。

正規化されたベクトルは，半径1の円周に沿って移動することになります。弧を描くわけですね。これを利用すると，p_1からp_2に向かって滑らかに補間されるベクトルを生成することができるのです。

次に，クォータニオンの場合の補間ですが，実はまったく同じ計算式でこれを行うことができます。tの関数である各係数を，クォータニオンに対してスカラー乗算してやれば同じことが行えます。

実際に使用する際には，パラメータとして存在するθはクォータニオン同士，つまり4次元ベクトルでの内積から求めることができます。注意すべきこととして，角度θが0度や180度に近い場合，つまり$\sin\theta$が0になってしまいそうな際には0除算を回避する策を用意しておかなければいけません。また，180度を超える角度の補間は行えません。日本から南極経由で北極に行きたくても，飛行機は勝手に北上してしまうのです。

まとめ

今回は，3Dプログラミングに登場するクォータニオンにまつわる話を網羅してみました。実際に使用するにはまだまだ説明の足りない部分も多いのですが，それでも読み始める前と比べると，クォータニオンに対する印象がずいぶん変わったのではないでしょうか。

さて，次回はまたまったく違った分野のことを取り上げてみたいと思います。以前光の話をしたときに「まじめに計算しようなんてどうかしている」と表現した話題，ラジオシティの登場です。