Utilização de quatérnios para representação de rotações em 3D

por

Sergio Coutinho de Biasi e Marcelo Gattass

Índice

1 Representação de rotações em 3D sem o uso de quatérnios

Quando desejamos modelar uma entidade (seja ela um corpo rígido, uma partícula, uma câmera, um raio de luz, etc…) precisamos, em muitas circunstâncias, especificar sua posição e orientação em nosso universo virtual.

A especificação de posições, usualmente dadas como translações com relação a uma origem conhecida, não apresenta grandes problemas. Grande parte das vezes podemos até mesmo simplesmente especificar a posição em coordenadas cartesianas e a questão está resolvida.

Já quanto às orientações, usualmente dadas como rotações com relação a uma origem inicial, não é assim tão simples. À primeira vista, especialmente aos iniciantes na área, pode parece que não haja qualquer dificuldade. Afinal, assim como no caso da posição, temos três graus de liberdade, e portanto bastariam três parâmetros para definir a orientação de uma entidade. A questão é : qual sistema de parametrizacão utilizar? Será que existe algum sistema adequado que use somente três parâmetros?

Neste trabalho, procuraremos apresentar as dificuldades envolvidas em representar rotações e mostraremos que uma das soluções mais práticas e intuitivas para este problema usa quatro parâmetros e se baseia na definição do grupo matemático dos assim chamados ***quatérnios***.

1.1 Ângulos de Euler

A solução mais imediata para o problema de especificar a orientação de uma entidade no espaço tridimensional é fornecer suas rotações com relação aos eixos x, y, e z. Inicialmente, parece que isso resolve toda a questão. Porém, há alguns problemas com essa solução.

Para sermos mais precisos, a parametrização proposta aqui é a seguinte : para especificar a orientação de uma entidade, forneceremos três parâmetros, que representam os ângulos de rotação anti-horária em relação a cada um dos três eixos coordenados. Esses ângulos são chamados de ângulos de Euler. Será que com isso o problema não está resolvido?

A resposta, um pouco surpreendente quando começamos a estudar o assunto, é que para muitas aplicações, esta representação é extremamente problemática.

1.1.1 Rotações tridimensionais não comutam

A primeira dificuldade está no fato de que operações de rotação, ao contário das de translação, não são comutativas. Ou seja, podemos representar a posição de um objeto como a soma dos deslocamentos paralelos a cada um dos eixos coordenados e ao final, não importando a ordem em que aplicarmos os três deslocamentos, terminaremos na mesma posição. Já com rotações, isso não é verdade. Se temos uma rotação em torno do eixo x e outra em torno do eixo y, e as aplicamos à nossa entidade, obteremos orientações finais diferentes dependendo da ordem em que as rotações forem aplicadas.

Caso o leitor nunca tenha se dado conta do fato que acabamos de enunciar, é importante que pare alguns momentos para compreender o significado do que foi afirmado. Descreveremos abaixo uma situação em que isso acontece.

Imagine o leitor que esteja no comando de um avião que voa em linha reta para a frente, indo para o norte, com a asa direita apontando para o leste e a esquerda para o oeste. Imagine então que o leitor gire o avião de 90 graus para a esquerda (isto é, uma rotação anti-horária em torno do eixo vertical), voando agora portanto para o oeste. Em seguida, imaginemos que o leitor continue a voar para o oeste mas incline o avião de forma a baixar sua asa direita e erguer a esquerda de 90 graus (isto é, uma rotação anti-horária em torno do eixo leste/oeste). Ao final, teremos o avião voando para o oeste, com a asa direita apontando para o solo e a esquerda para o céu.

Verifiquemos agora o que ocorre se executarmos exatamente as mesmas duas rotações, porém na ordem inversa. Começando com o avião voando para o norte, se executarmos a rotação anti-horária de 90 graus em torno do eixo leste/oeste, o avião passará a voar na vertical, com a cabine voltada para o solo, com a cauda apontando para o céu e a barriga para o sul. A asa direita continuará apontando para o leste e a esquerda para o oeste. Se agora executarmos a rotação anti-horária de 90 graus em torno do eixo vertical, teremos o avião ainda voando para baixo, mas com a asa direita apontando para o norte, a esquerda para o sul e a barriga voltada para o leste.

Portanto, como se pode ver no exemplo acima, a ordem em que executamos as rotações pode alterar completamente a orientação final obtida. Isso significa que para descrever a orientação de uma entidade, não é suficiente fornecer os ângulos de rotação em torno dos eixos coordenados; é preciso também especificar a ***ordem*** em que essas rotações devem ser executadas.

Mais uma vez, talvez pareça ao leitor que não há qualquer problema. Num primeiro momento, parece razoável pensar que bastaria especificarmos uma ordem padronizada para as rotações. Assim, para especificarmos a orientação de uma entidade, forneceríamos os ângulos de rotação em torno dos eixos coordenados e

estaria subentendido que primeiro devemos executar a rotação em torno de *x*, em seguida em torno de *y* e finalmente em torno de *z*, necessariamente nesta ordem.

1.1.2 Representação de orientações fixas

Examinemos, portanto, o que acontece se representamos a orientação de uma entidade através de três ângulos representando rotações anti-horárias numa ordem predeterminada em torno dos eixos coordenados.

Voltando ao nosso avião, imaginemos que, após ler a seção anterior, tenhamos resolvido fixar a ordem das rotações como sendo sucessivamente em torno dos eixos sul/norte, leste/oeste e baixo/cima. (A discussão seria exatamente a mesma se tivéssemos chamado nossos eixos de *x*, *y* e *z*. Se o leitor preferir, pode pensar, nos exemplos que se seguem, no eixo sul/norte como sendo o eixo *x*, no leste/oeste como *y* e no baixo/cima como *z*).

Neste sistema de representação, para especificar a orientação inicial "voando para o norte com a asa direita apontando para o leste" forneceríamos três ângulos : $(\theta_1,\theta_2,\theta_3) = (0,0,0)$. Essa seria a "origem" do nosso sistema de representação de orientações, assim como o ponto (x,y,z) = (0,0,0) seria a origem de um sistema de representação de posições.

Para especificar a orientação obtida ao final da primeira parte de nosso exemplo, ou seja, voando para o oeste com a asa direita apontando para o solo, não poderíamos usar os ângulos do exemplo, pois nele executamos primeiro a rotação em torno do eixo vertical e em seguida a rotação em torno do eixo leste/oeste, contrariando a ordem que escolhemos. Porém, se primeiro inclinarmos o avião para a direita (rotação em torno do eixo sul/norte) para então apontá-lo para o oeste (rotação em torno do eixo vertical) obteremos o mesmo resultado. Portanto, para representar essa orientação usando a ordem que escolhemos, forneceríamos os ângulos (90,0,90). O leitor deve verificar que esta seqüência de rotações efetivamente gera a mesma orientação final que a obtida antes.

Finalmente, para especificar a orientação obtida ao final da segunda parte do exemplo, isto é, voando para baixo com a asa direita apontando para o norte, não há qualquer dificuldade, pois as rotações já estão na ordem correta, e simplesmente usaríamos os ângulos (0,90,90).

Por enquanto, parece que tudo vai muito bem, e de fato podemos, sem grandes dificuldades, representar qualquer orientação fixa utilizando este sistema. Se fosse este nosso único objetivo, nosso problema provavelmente estaria resolvido de forma satisfatória.

1.1.3 Representação de orientações mutáveis

Em muitas situações, porém, desejamos representar orientações que estão continuamente sofrendo pequenas alterações. Ao animarmos o movimento de um avião, por exemplo, normalmente não desejamos saltar repentinamente entre orientações fixas predeterminadas e sim alterar pouco a pouco a orientação do avião, seja devido a uma pequena correção da rota, seja para executar de forma suave uma grande correção da rota.

Neste tipo de contexto, a utilização de ângulos de Euler apresenta duas dificuldades principais, que examinaremos a seguir.

1.1.3.1 Gimbal lock

Gimbal lock é o nome dado a um fenômeno não muito intuitivo (mas muito real) com o qual se defrontam animadores que representam a orientação das entidades em seu universo virtual utilizando ângulos de Euler.

Ao invés de descrever o problema conceitualmente, comecemos por mostrar que ele existe através de um exemplo.

Imaginemos que estamos modelando um avião voando inicialmente para o norte, com a asa direita voltada para o leste. Suponhamos que o piloto comece a baixar o nariz do avião aos poucos. Em nossa representação, isso significará uma rotação anti-horária em torno do eixo leste/oeste, ou seja, o ângulo correspondente começará a crescer. Se o piloto baixar o nariz do avião até que esteja voando diretamente de encontro ao solo, ele terá executado uma rotação de 90 graus, e portanto sua orientação neste momento será representada por (0,90,0) e a barriga do avião estará voltada para o sul.

Suponhamos agora que o piloto decida executar um parafuso, ou seja, girar o avião em torno de seu eixo longitudinal. Como o avião está indo para baixo, isso significa girar em torno do eixo vertical, ou seja, se ele girar no sentido anti-horário, a asa esquerda girará para o sul e a direita para o norte. Para que esta rotação ocorra suavemente, o ângulo de rotação vertical terá que aumentar aos poucos, até completar a rotação total pretendida. Até aqui, não temos qualquer problema : após um giro de 90 graus, o avião estará na orientação (0,90,90), com a barriga para o leste, após 180 graus em (0,90,180), com a barriga para o norte e, finalmente, após uma volta completa, terá voltado a (0,90,0), com a barriga novamente voltada para o sul.

Digamos, porém, que uma vez indo para baixo, com a barriga para o sul, o piloto decida, ao invés de executar um parafuso, girar o avião para a sua esquerda,

apontando a asa esquerda para o céu, ou seja, uma rotação anti-horária em torno do eixo sul/norte. Como executar esta rotação de forma suave, incremental?

Inicialmente, parece razoável supor que basta incrementar aos poucos o ângulo correspondente à rotação sul/norte. Só que isso não funciona, pois a rotação em torno do eixo sul/norte é, em nossa convenção, executada ***antes*** da rotação em torno do exito leste/oeste. Surpreendentemente, se incrementarmos a rotação sul/norte ***também*** executaremos um parafuso. O leitor deve verificar que é realmente esse o caso.

O fato é que, enquanto mantivermos a rotação de 90 graus ao redor do eixo leste/oeste, não há qualquer mudança nos outros eixos que possa nos levar a alterações na orientação em torno do eixo sul/norte. Isso não significa que a nova posição desejada não tenha representação através de nenhuma combinação de ângulos de Euler; porém, enquanto o ângulo de rotação leste/oeste permanecer em 90 graus, efetivamente perdemos um grau de liberdade de movimento e há orientações que nunca poderemos atingir. Essa é a situação chamada de ***gimbal lock***.

1.1.3.2 Evitando o Gimbal lock

Ou seja, uma vez que tenhamos nossa entidade em uma determinada orientação, se desejamos girá-la um pouco mais, mesmo que seja em torno de um dos três eixos coordenados e não de um eixo arbitrário, não basta simplesmente incrementar um pouco a rotação do eixo correspondente, pois queremos executar a nova rotação ***após*** as rotações que já foram previamente executadas.

Levando isso em conta, uma solução seria simplesmente guardar uma lista de todas as rotações executadas sobre a entidade, na exata ordem em que foram executadas. Porém, isso significaria guardar uma quantidade cada vez maior de dados, e repetir toda a história de rotações cada vez que quiséssemos orientar a entidade. Confuso, ineficiente, e muito pouco prático.

Outra solução seria representar diretamente a orientação através de uma matriz de rotação com relação à posição inicial e simplesmente multiplicar essa matriz por cada nova rotação aplicada à entidade. Em princípio, isso funcionaria, e não exigiria o armazenamento de toda a história de rotação, que estaria condensada em uma só matriz. Essa solução, porém, também apresenta problemas. Em primeiro lugar, estamos usando uma matriz 3x3 para representar algo que só tem três graus de liberdade – ou seja, com certeza estamos guardando desnecessariamente informação redundante. Pior do que isso, as sucessivas multiplicações executadas sobre a matriz inevitavelmente acumulam erros, fazendo com que a orientação final se distancie da pretendida ou, pior ainda, a transformação representada pela

matriz pode até mesmo deixar de ser uma rotação, distorcendo a entidade representada. Este último problema pode ser resolvido renormalizando-se periodicamente a matriz que representa a orientação, mas isto acrescenta ainda mais custo, complexidade e fontes de imprecisão.

Finalmente, poderíamos, a cada pequena rotação, recalcular os três ângulos de Euler que representam essa rotação. O problema com essa solução é que os novos ângulos de Euler não estarão necessariamente relacionados de nenhuma forma óbvia com os ângulos antes da rotação. Como já vimos antes, às vezes para girar em torno de um eixo precisamos executar uma outra rotação prévia em torno de outro eixo. Próximo aos pontos de gimbal lock haveria saltos ainda menos óbvios. O que precisamos é de um sistema de representação em que operações como "gire ao redor do eixo tal" possam ser executadas de forma natural e automática.

1.1.3.3 O problema da interpolação de orientações

A segunda dificuldade importante apresentada pela representação através de ângulos de Euler surge quando desejarmos interpolar entre duas orientações, isto é, produzir uma seqüência de orientações intermediárias entre duas orientações dadas.

Mesmo que não se incorra no problema de gimbal lock, ainda assim não é óbvio como fazer com que uma entidade execute uma transição suave entre duas orientações.

Novamente, temos aqui uma situação bem diferente do caso da translação simples, na qual a interpolação, pelo menos no caso mais trivial, é imediata. Se desejamos que uma entidade se mova de forma suave entre duas posições percorrendo uma linha reta, simplesmente interpolamos linearmente cada uma de suas coordenadas, de forma independente, e o problema está resolvido – teremos produzidos tantas posições intermediárias quantas quisermos ao longo da linha reta que liga as duas posições.

No caso de uma mudança de orientação, se estivermos trabalhando com ângulos de Euler, no entanto, essa solução não gera resultados muito satisfatórios. A interpolação aplicada sobre cada um dos ângulos de rotação gerará rotações independentes em torno desses eixos, ao invés de uma rotação suave e natural em torno do eixo desejado.

Como exemplo, consideremos o caso de nosso avião imaginário. Suponhamos que, ao realizar uma animação de seu movimento, desejemos que ele execute uma rotação da orientação (0,0,0) até a orientação (0,180,180).

A primeira orientação é aquela que já conhecemos, do avião voando de cabeça para cima para o norte.

A segunda, pode-se concluir facilmente, corresponde ao avião voando de cabeça para baixo para o norte. Basta verificar que a rotação de 180 graus em torno do eixo leste/oeste deixa o avião voando de cabeça para baixo em direção ao sul, e a posterior rotação em torno do eixo vertical o coloca voando de novo em direção ao norte, mas ainda de cabeça para baixo.

Se simplesmente utilizarmos uma interpolação linear entre (0,0,0) e (0,180,180), produziremos orientações intermediárias que parecerão muito pouco naturais. Para virar de cabeça para baixo, o avião poderia simplesmente girar 180 graus em torno do eixo sul/norte. Com a interpolação linear de (0,0,0) a (0,180,180), entretanto, ele executará uma cambalhota estranha na qual girará simultaneamente em torno dos eixos leste/oeste e do vertical. A orientação final será a mesma, mas o movimento intermediário não. O leitor deve procurar compreender a diferença entre as duas formas descritas de interpolar entre as duas orientações.

Transições entre outros pares de orientações utilizando interpolação linear dos ângulos de Euler em geral gerarão movimentos igualmente estranhos e imprevisíveis.

1.1.3.4 A interpolação "natural" entre orientações

Isso nos leva à questão : afinal, qual seria a forma "natural" de determinar posições intermediárias entre duas orientações? Em outras palavras, qual o "caminho" que uma entidade deve seguir para transicionar suavemente de uma orientação para outra?

Para colocar melhor a questão, examinemos o caso da interpolação entre posições. Se desejamos que uma entidade se mova suavemente de uma posição para outra, devemos determinar uma seqüência de posições intermediárias entre as posições inicial e final. Porém, dadas duas posições no espaço tridimensional, há uma infinidade de curvas que as ligam. A entidade poderia mover-se de uma posição a outra em zigue-zague, ou passando primeiro por uma outra posição longínqua, ou através de outros caminhos arbitrariamente convolutos. No caso da translação, a solução mais simples e imediata para o problema é percorrer simplesmente uma linha reta, sem que a entidade execute quaisquer desvios "desnecessários".

No caso da transição entre duas orientações, desejamos, em princípio, algo semelhante, ou seja, que a transição não inclua voltas e cambalhotas que nos pareçam "desnecessárias" para chegar à orientação final. Como formalizar esse conceito?

Felizmente, há uma solução bastante natural para a questão. Ela surge do fato (demonstrado por Euler) que ***sempre*** é possível chegar de uma orientação a outra

através de uma rotação simples, ao redor de um único eixo. Logo, dadas duas orientações, basta executarmos uma interpolação linear simples no ângulo de rotação em torno desse eixo, que sabemos que necessariamente existe, para obtermos uma transição suave, única (fora o sentido de rotação) e sem "desvios".

Esse eixo, porém, não é necessariamente um dos eixos coordenados, e a parametrização da orientação através de ângulos de Euler não leva, de forma prática ou natural, à realização de rotação ao redor de eixos arbitrários. Seria desejável encontrar uma parametrização na qual a transição entre duas orientações ocorresse naturalmente ao redor do eixo adequado e não seguindo um caminho arbitrário.

1.2 Rotação ao redor de um eixo

A forma mais "natural" de expressar tanto orientações como rotações arbitrárias seria, portanto, a especificação de um eixo e de um ângulo de rotação. Porém, como fazer para, a partir de um ponto no espaço, um eixo dado e um ângulo de rotação, determinar a nova posição do ponto após sofrer a rotação especificada?

De forma mais precisa, seja um ponto no $\Re^3$ representado por um vetor $\vec{r} = (r_x, r_y, r_z)$. Seja $\rho_{\theta,\vec{n}}$ uma rotação anti-horária de um ângulo θ em torno de um eixo que intercepta a origem definido por um vetor unitário $\vec{n} = (n_x, n_y, n_z)$. Desejamos determinar uma expressão para $\rho(\vec{r})$, ou seja, para o vetor que representa o ponto obtido após aplicarmos a $\vec{r}$ a rotação ρ.

O problema pode ser resolvido decompondo $\vec{r}$ em suas componentes normal $\vec{r}_\perp$ e paralela $\vec{r}_\parallel$ ao vetor $\vec{n}$, aplicando a rotação separadamente a essas componentes e somando os resultados. Para obtermos a magnitude da componente de $\vec{r}$ paralela ao vetor $\vec{n}$, basta realizarmos o produto escalar entre $\vec{n}$ e $\vec{r}$.

Pensando assim, obtemos que

$$\begin{aligned} \vec{r} &= \vec{r}_\parallel + \vec{r}_\perp \\ \vec{r}_\parallel &= (\vec{n}\cdot\vec{r})\vec{n} \\ \vec{r}_\perp &= \vec{r} - \vec{r}_\parallel = \vec{r} - (\vec{n}\cdot\vec{r})\vec{n} \end{aligned} \qquad \textbf{(1)}$$

A componente $\vec{r}_\parallel$, naturalmente, permanece inalterada por uma rotação em torno do eixo definido por $\vec{n}$, de forma que temos

$$\rho(\vec{r}_\parallel) = \vec{r}_\parallel \qquad \textbf{(2)}$$

Portanto, o problema que nos resta é determinar qual o resultado da rotação de $\vec{r}_\perp$. Sabemos que, por definição esta rotação ocorrerá num plano paralelo a $\vec{r}_\perp$ e perpendicular a $\vec{n}$. Logo, se definirmos o vetor $\vec{v}$ como

$$\vec{v} = \vec{n} \times \vec{r}_\perp = \vec{n} \times (\vec{r} - \vec{r}_\parallel) = \vec{n} \times \vec{r} - \vec{n} \times \vec{r}_\parallel = \vec{n} \times \vec{r} \qquad \textbf{(3)}$$

teremos que $\vec{n}$, $\vec{r}_\perp$ e $\vec{v}$ formarão um triedro direto, e em particular que $\vec{r}_\perp$ e $\vec{v}$ serão perpendiculares e estarão no plano onde ocorrerá a rotação. Além disso, como $\vec{n}$ é unitário, $\vec{v}$ terá a mesma norma que $\vec{r}_\perp$. Portanto, é imediato verificar que **(4)**

$$\rho(\vec{r}_\perp) = (\cos\theta)\vec{r}_\perp + (\text{sen}\theta)\vec{v}$$

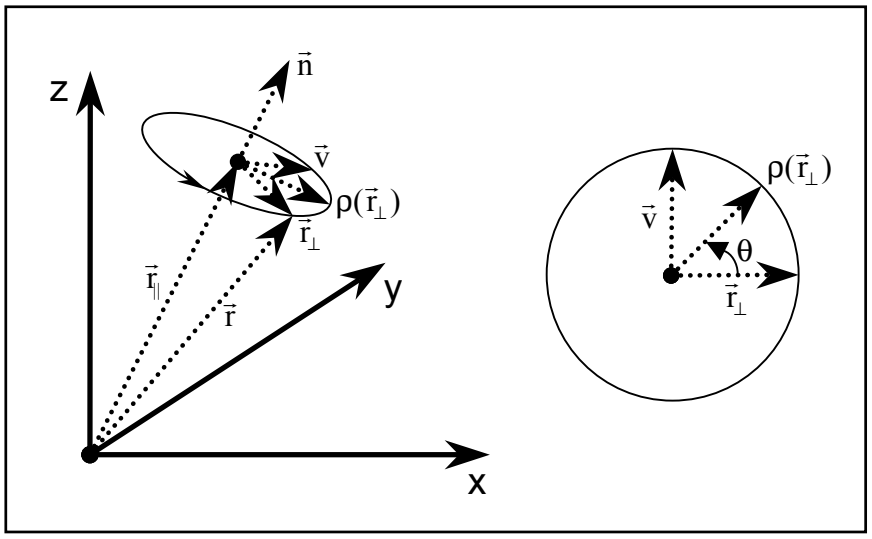

Figura 1 : Rotação em torno de um eixo

Somando as componentes, encontramos que

$$\begin{aligned}
\rho(\vec{r}) &= \rho(\vec{r}_\parallel + \vec{r}_\perp) \\
&= \rho(\vec{r}_\parallel) + \rho(\vec{r}_\perp) \\
&= \vec{r}_\parallel + (\cos\theta)\vec{r}_\perp + (\text{sen}\theta)\vec{v} \\
&= (\vec{n}\cdot\vec{r})\vec{n} + (\cos\theta)(\vec{r} - (\vec{n}\cdot\vec{r})\vec{n}) + (\text{sen}\theta)(\vec{n}\times\vec{r}) \\
&= (\vec{n}\cdot\vec{r})\vec{n} + (\cos\theta)\vec{r} - (\cos\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}\theta)(\vec{n}\times\vec{r}) \\
&= (\cos\theta)\vec{r} + (1-\cos\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}\theta)(\vec{n}\times\vec{r})
\end{aligned} \qquad \textbf{(5)}$$

Desta forma, concluímos que o ponto resultante da realização de uma rotação $\rho(\theta, \vec{n})$ em um ponto $\vec{r}$ é

$$\rho(\vec{r}) = (\cos\theta)\vec{r} + (1-\cos\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}\theta)(\vec{n}\times\vec{r}) \quad \textbf{(6)}$$

2 Representação de rotações em 3D com o uso de quatérnios

Como se pôde observar, a realização de rotações em torno de eixos arbitrários leva a expressões extensas e pouco intuitivas se operarmos diretamente com eixos cartesianos e ângulos. Os quatérnios nos fornecem um sistema de representação bem mais adequado para operar sobre rotações.

2.1 Rotacões em 2D e números complexos

Sob vários aspectos, os quatérnios podem ser encarados como uma generalização, no espaço tridimensional, do que os números complexos representam para o espaço bidimensional. Portanto, começaremo por relembrar algumas características dos números complexos.

Um número complexo é definido através de dois parâmetros, números reais usualmente chamados de a e b. O parâmetro a é chamado de parte real do número complexo, e o parâmetro b, de parte imaginária. A unidade dos números imaginários, ou seja, o número complexo (0,1), é normalmente chamada de **i**. Portanto, outra forma de representar um número complexo é da forma

$$c = a+b\mathbf{i} \tag{7}$$

As propriedades de adição e multiplicação dos números complexos são definidas de tal forma que, entre outras propriedades importantes, eles podem ser utilizados para representar rotações no plano.

Como cada número complexo é dado por dois parâmetros, podemos interpretá-los como coordenadas cartesianas planas. Dessa forma, o parâmetro a corresponde à coordenada x e o parâmetro b à coordenada y. Podemos, portanto, pensar em um número complexo como um vetor no plano.

Uma outra forma de representação, que será importante ao operarmos com rotações, é a forma polar, na qual um vetor é descrito através de sua norma (comprimento cartesiano) e e de seu deslocamento angular (rotação anti-horária em torno da origem a partir do eixo dos x).

Em resumo, podemos pensar em um número complexo como uma soma algébrica de um número real com um número imaginário, na forma $a+b\mathbf{i}$, como um vetor cartesiano (a,b) ou como um vetor polar (r,θ).

A soma de dois números complexos é dada simplesmente pela soma vetorial cartesiana, componente a componente, com a qual estamos acostumados, ou seja :

$$a_1+b_1\mathbf{i} + a_2+b_2\mathbf{i} = a_1+a_2 + (b_1+b_2)\mathbf{i} \quad \textbf{(8)}$$

Isto é exatamente o mesmo que dizer que

$$(a_1, b_1) + (a_2, b_2) = (a_1+a_2, b_1+b_2) \quad \textbf{(9)}$$

Já a multiplicação, porém, tem a seguinte propriedade : quando multiplicamos dois números complexos, obtemos um vetor cuja norma é o produto das normas dos vetores sendo multiplicados e cujo deslocamento angular é a soma dos deslocamentos angulares de cada vetor. Em coordenadas polares, isso resulta simplesmente que : **(10)**

$$(r_1, \theta_1)\,(r_2, \theta_2) = (r_1r_2, \theta_1+\theta_2)$$

A propriedade dos deslocamentos angulares se somarem significa que podemos pensar na multiplicação entre dois complexos como uma operação de rotação. No caso de um complexo com norma unitária, teremos a representação de uma rotação pura, que numa operação de multiplicação alterará apenas a orientação de um outro vetor sem modificar sua norma. Em particular, o imaginário puro **i**, que corresponde ao vetor polar (1,90), representará sempre uma rotação anti-horária de 90 graus em torno da origem.

Examinemos um exemplo concreto. Se multiplicamos o vetor cartesiano (1,1), que corresponde ao vetor polar ($\sqrt{2}$,45), pelo vetor cartesiano (0,1), que corresponde ao vetor polar (1,90), obteremos, pelo enunciado acima, o vetor polar ($\sqrt{2}$,45)(1,90) = ($\sqrt{2}$,90+45) = ($\sqrt{2}$,135), que corresponde ao vetor cartesiano (-1,1). Este resultado pode ser pode ser interpretado tanto como o vetor cartesiano (1,1) rotacionado de 90 graus no sentido anti-horário quanto como o vetor (1,0) escalado por um fator de $\sqrt{2}$ e rotacionado de 45 graus no sentido anti-horário.

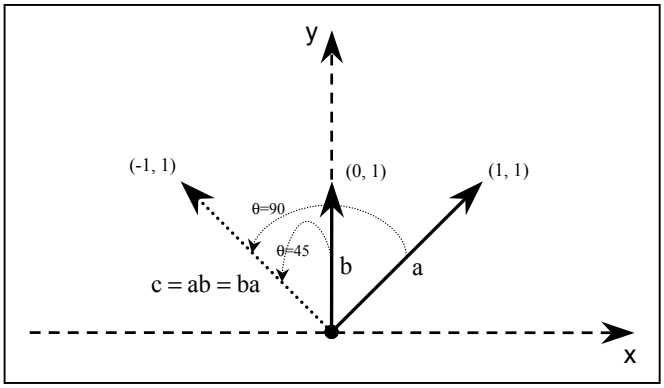

Figura 2 : Rotação com complexos

De fato, é fácil verificar que como tanto a multiplicação de normas quanto a soma de ângulos de rotação plana são comutativas, então a multiplicação de números complexos é, conseqüentemente, sempre comutativa, como pudemos observar no exemplo acima.

Como realizar tais operações de forma algébrica, operando diretamente sobre as coordenadas cartesianas?

Se realizarmos a multiplicação entre dois complexos da forma algébrica usual, obteremos que

$$\begin{aligned}&(a_1+b_1\mathbf{i})(a_2+b_2\mathbf{i}) = \\ &a_1a_2 + a_1b_2\mathbf{i} + b_1\mathbf{i}a_2 + b_1\mathbf{i}b_2\mathbf{i} = \\ &a_1a_2 + (a_1b_2 + a_2b_1)\mathbf{i} + b_1b_2\mathbf{i}^2\end{aligned} \qquad \textbf{(11)}$$

A questão é : o que fazer com o termo $\mathbf{i}^2$ que surgiu? Qual o seu significado?

Como queremos fazer valer as propriedades previamente enunciadas para a multiplicação, então somos obrigados a concluir que $\mathbf{i}^2$ deverá calculado como a multiplicação do vetor cartesiano (0,1) por si mesmo. Isso significa multiplicar o vetor polar (1,90) por si mesmo, cujo resultado é (1,180). Voltando às coordenadas cartesianas, verificamos que obtivemos o vetor (-1,0). Em outras palavras, o número **i,** ao operar uma rotação de 90 graus sobre o vetor (0,1), produz o vetor (-1,0). Ou seja, concluímos que **(12)**

$$\mathbf{i}^2 = -1$$

Com base nisso, podemos agora escrever a expressão final para a multiplicação : **(13)**

$$(a_1+b_1\mathbf{i})(a_2+b_2\mathbf{i}) = (a_1a_2-b_1b_2) + (a_1b_2 + a_2b_1)\mathbf{i}$$

Com estas definições para as operações envolvendo números complexos, podemos representar e operar sobre orientações e rotações bidimensionais de forma prática e automática.

2.2 Generalizando os números complexos

Apresentada esta solução para o problema de rotação bidimensional, pode parecer que não seria muito difícil construir um sistema semelhante para o tratamento de rotações tridimensionais.

Infelizmente, porém, a generalização dos complexos para o espaço tridimensional não é tão imediata e exigiu muito esforço até ser concebida.

Talvez um dos principais impecilhos tenha sido o fato de que parece intuitivo imaginar que uma tal generalização se basearia em três parâmetros, ou seja, no acréscimo de um parâmetro extra para a dimensão extra acrescentada.

No entanto, embora apenas um grau de liberdade tenha sido acrescentado à translação, o impacto disso sobre as possibilidades de rotação é bem mais forte.

O fato é que, num mundo bidimensional, existe apenas um grau de liberdade de orientação, isto é, o que corresponde à rotação em torno da origem. Ao parametrizar uma rotação, precisamos especificar apenas seu ângulo, pois o eixo é obrigatoriamente o perpendicular ao plano considerado. Portanto, quando acrescentamos a terceira dimensão, ganhamos não um mas dois graus de liberdade de rotação, que precisam, para serem representados de forma similar à dos complexos, de uma unidade imaginária cada um.

O resultado é que a generalização dos complexos para o mundo tridimensional usa naturalmente não três mas quatro parâmetros e é correspondetemente constituída por número chamados de ***quatérnios***.

2.3 Quatérnios

Os quatro parâmetros que definem um quatérnio são números reais usualmente chamados de a, b, c, e d.

De forma similar aos números complexos, os quatérnios possuem uma parte real e uma imaginária. Ao contrário dos complexos, porém, os quatérnios apresentam três componentes diferentes para sua parte imaginária, às quais chamaremos de **i**, **j** e **k**.

Um quatérnio, portanto, é um número da forma q = (a,b,c,d) ou, equivalentemente,

$$q = a+b\mathbf{i}+c\mathbf{j}+d\mathbf{k} \qquad \textbf{(13)}$$

Podemos também expressar um quatérnio de forma ainda mais condensada dizendo que $q = (s, \vec{v})$, onde s é um escalar que representa sua parte real e $\vec{v} = (v_x, v_y, v_z)$ é um vetor de três componentes que representa sua parte imaginária.

As propriedades de **i**, **j** e **k** são generalizações das propriedades do **i** dos complexos. As definições que levam às propriedades desejadas exigiram, historicamente, muita reflexão antes de serem adequadamente formuladas, e são apresentadas a seguir.

De forma similar aos complexos, temos que

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1 \qquad \textbf{(14)}$$

Porém, ao contrário dos complexos, precisamos lidar também com multiplicações entre imaginários de natureza diferente. Isso é resolvido definindo que

$$\mathbf{ij} = \mathbf{k}$$
$$\mathbf{ji} = -\mathbf{k} \qquad \textbf{(15)}$$

Da definição acima, é fácil verificar que

$$\mathbf{ij} = \mathbf{k} \Rightarrow \mathbf{ijk} = \mathbf{kk} \Rightarrow \mathbf{ijk} = -1 \Rightarrow \mathbf{iijk} = -\mathbf{i} \Rightarrow -\mathbf{jk} = -\mathbf{i} \Rightarrow \mathbf{jk} = \mathbf{i} \qquad \textbf{(16)}$$

Assim como que

$$\mathbf{jk} = \mathbf{i} \Rightarrow \mathbf{jki} = \mathbf{ii} \Rightarrow \mathbf{jki} = -1 \Rightarrow \mathbf{jjki} = -\mathbf{j} \Rightarrow -\mathbf{ki} = -\mathbf{j} \Rightarrow \mathbf{ki} = \mathbf{j} \qquad \textbf{(17)}$$

Observe-se que o produto de dois quatérnios, ao contrário dos complexos, não é comutativo, como aliás seria de se esperar, já que as rotações tridimensionais, ao contrário das planas, não comutam.

A soma de dois quatérnios é simplesmente a soma algébrica usual componente a componente.

A relevância dos quatérnios reside em sua operação de multiplicação. A partir das definições acima, podemos verificar que, dados dois quatérnios q_1 e q_2, seu produto será :

$$\begin{aligned}
q_1q_2 &= (a_1+b_1\mathbf{i}+c_1\mathbf{j}+d_1\mathbf{k})(a_2+b_2\mathbf{i}+c_2\mathbf{j}+d_2\mathbf{k}) = \\
&= a_1(a_2+b_2\mathbf{i}+c_2\mathbf{j}+d_2\mathbf{k}) + b_1\mathbf{i}(a_2+b_2\mathbf{i}+c_2\mathbf{j}+d_2\mathbf{k}) + \\
&c_1\mathbf{j}(a_2+b_2\mathbf{i}+c_2\mathbf{j}+d_2\mathbf{k}) + d_1\mathbf{k}(a_2+b_2\mathbf{i}+c_2\mathbf{j}+d_2\mathbf{k}) = \\
&= a_1a_2+ a_1b_2\mathbf{i}+ a_1c_2\mathbf{j}+ a_1d_2\mathbf{k} + b_1\mathbf{i}a_2+ b_1\mathbf{i}b_2\mathbf{i}+ b_1\mathbf{i}c_2\mathbf{j}+ b_1\mathbf{i}d_2\mathbf{k} + \\
&c_1\mathbf{j}a_2+ c_1\mathbf{j}b_2\mathbf{i}+ c_1\mathbf{j}c_2\mathbf{j}+ c_1\mathbf{j}d_2\mathbf{k} + d_1\mathbf{k}a_2+ d_1\mathbf{k}b_2\mathbf{i} + d_1\mathbf{k}c_2\mathbf{j}+ d_1\mathbf{k}d_2\mathbf{k} = \\
&= a_1a_2+ b_1b_2\mathbf{i}^2+ c_1c_2\mathbf{j}^2+ d_1d_2\mathbf{k}^2 + a_1b_2\mathbf{i} + a_1c_2\mathbf{j}+ a_1d_2\mathbf{k} + b_1a_2\mathbf{i} + \\
&c_1a_2\mathbf{j} + d_1a_2\mathbf{k}+ b_1c_2\mathbf{ij}+ b_1d_2\mathbf{ik} + c_1b_2\mathbf{ji}+ c_1d_2\mathbf{jk} + d_1b_2\mathbf{ki} + d_1c_2\mathbf{kj} = \\
&= a_1a_2 - (b_1b_2 + c_1c_2 + d_1d_2) + a_1(b_2\mathbf{i} + c_2\mathbf{j}+ d_2\mathbf{k}) + a_2(b_1\mathbf{i} + c_1\mathbf{j} + \\
&d_1\mathbf{k}) + b_1c_2\mathbf{k} - b_1d_2\mathbf{j} - c_1b_2\mathbf{k}+ c_1d_2\mathbf{i} + d_1b_2\mathbf{j} - d_1c_2\mathbf{i} = \\
&= a_1a_2 - (b_1b_2 + c_1c_2 + d_1d_2) + a_1(b_2\mathbf{i} + c_2\mathbf{j}+ d_2\mathbf{k}) + a_2(b_1\mathbf{i} + c_1\mathbf{j} + \\
&d_1\mathbf{k}) + (c_1d_2 - d_1c_2)\mathbf{i} + (d_1b_2 - b_1d_2)\mathbf{j} + (b_1c_2 - c_1b_2)\mathbf{k}
\end{aligned} \qquad \textbf{(18)}$$

Coletando os termos obtidos acima e expressando o resultado em termos de produtos escalares e vetoriais, podermos escrever esta fórmula como

$$q_1q_2 = (s_1, \vec{v}_1)(s_2, \vec{v}_2) = (s_1s_2 - \vec{v}_1 \cdot \vec{v}_2,\ s_1\vec{v}_2 + s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2) \qquad \textbf{(19)}$$

O conjugado de um quatérnio $q = (s, \vec{v})$ é representado por $\overline{q}$ e definido como

$$\overline{q} = (s, -\vec{v}) \qquad \textbf{(20)}$$

O inverso de um quatérnio q é representado por q^{-1} e definido como o quatérnio tal que

$$qq^{-1} = q^{-1}q = 1 \qquad \textbf{(21)}$$

A magnitude de um quatérnio é obtida através do produto do quatérnio por seu conjugado :

$$\begin{aligned}
|q|^2 &= q\overline{q} = (s, \vec{v})(s, -\vec{v}) = (s^2 - \vec{v} \cdot -\vec{v},\ -s\vec{v} + s\vec{v} + \vec{v} \times -\vec{v}) = \\
&= (s^2 + \vec{v} \cdot \vec{v},\ 0) = s^2 + |\vec{v}|^2
\end{aligned} \qquad \textbf{(22)}$$

2.3.1 Representação e composição de rotações

Dadas essas definições, passemos agora a descrever como os quatérnios podem ser usados para representar rotações.

O ponto $\vec{r} = (r_x, r_y, r_z)$ sobre o qual queremos executar uma rotação será representado por um quatérnio $p = (0, \vec{r})$ com parte real nula.

A rotação que aplicaremos sobre o ponto $\vec{r}$ será representada por um quatérnio unitário $q = (s, \vec{v})$, isto é, um quatérnio q tal que $q\overline{q} = 1$.

Passaremos agora a demonstrar que o resultado da rotação de p por q poderá ser obtido através da expressão

$$R_q(p) = qpq^{-1} \quad \textbf{(23)}$$

Como q é unitário, temos que o inverso de q é seu conjugado, pois

$$q\overline{q} = 1 \Rightarrow q^{-1}q\overline{q} = q^{-1} \Rightarrow \overline{q} = q^{-1} \quad \textbf{(24)}$$

Logo, a expressão de rotação pode ser escrita como

$$R_q(p) = qp\overline{q} \quad \textbf{(25)}$$

Expandindo esta expressão, obtemos :

$$\begin{aligned}
&qp\overline{q} = \\
&(s, \vec{v})(0, \vec{r})(s, -\vec{v}) = \\
&(s, \vec{v})(0\,s - \vec{r}\cdot -\vec{v},\ -0\,\vec{v} + s\vec{r} + \vec{r}\times -\vec{v}) = \\
&(s, \vec{v})(\vec{r}\cdot\vec{v},\ s\vec{r} - \vec{r}\times\vec{v}) = \\
&(s(\vec{r}\cdot\vec{v}) - \vec{v}\cdot(s\vec{r} - \vec{r}\times\vec{v}),\ s(s\vec{r} - \vec{r}\times\vec{v}) + (\vec{r}\cdot\vec{v})\vec{v} + \vec{v}\times(s\vec{r} - \vec{r}\times\vec{v})) = \\
&(s(\vec{r}\cdot\vec{v}) - \vec{v}\cdot s\vec{r} - \vec{v}\cdot(-\vec{r}\times\vec{v}),\ s^2\vec{r} - s\vec{r}\times\vec{v} + (\vec{r}\cdot\vec{v})\vec{v} + \vec{v}\times s\vec{r} + \vec{v}\times(-\vec{r}\times\vec{v}) = \\
&(s(\vec{r}\cdot\vec{v}) - s(\vec{r}\cdot\vec{v}) + \vec{v}\cdot(\vec{r}\times\vec{v}),\ s^2\vec{r} + s\vec{v}\times\vec{r} + (\vec{r}\cdot\vec{v})\vec{v} + s\vec{v}\times\vec{r} + \vec{v}\times(\vec{v}\times\vec{r}) = \\
&(\vec{v}\cdot(\vec{r}\times\vec{v}),\ s^2\vec{r} + (\vec{r}\cdot\vec{v})\vec{v} + 2s\vec{v}\times\vec{r} + (\vec{v}\cdot\vec{r})\vec{v} - (\vec{v}\cdot\vec{v})\vec{r} = \\
&(0,\ s^2\vec{r} - (\vec{v}\cdot\vec{v})\vec{r} + 2(\vec{v}\cdot\vec{r})\vec{v} + 2s\vec{v}\times\vec{r})
\end{aligned} \quad \textbf{(24)}$$

Observação : na dedução acima, utilizamos as identidades a seguir :

$$\begin{aligned}
&\vec{a}\cdot(\vec{b}\times\vec{a}) = 0 \\
&\vec{a}\times\vec{b} = -\vec{b}\times\vec{a} \\
&\vec{a}\times(\vec{b}\times\vec{c}) = (\vec{a}\cdot\vec{c})\vec{b} - (\vec{a}\cdot\vec{b})\vec{c}
\end{aligned} \qquad \textbf{(25)}$$

Por outro lado, como $q = (s, \vec{v})$ é unitário, temos que $s^2 + |\vec{v}|^2 = 1$. Isso significa que sempre existe um ângulo θ tal que $s = \cos\theta$ e $|\vec{v}| = \text{sen}\theta$. Logo, sempre podemos escrever q como

$$q = (s, \vec{v}) = (\cos\theta,\ \text{sen}\theta\ \vec{n}), \qquad |\vec{n}| = 1 \qquad \textbf{(26)}$$

Se substituirmos agora esta interpretação de q na expressão obtida anteriormente para o resultado da rotação, obteremos

$$\begin{aligned}
&(0,\ s^2\vec{r} - (\vec{v}\cdot\vec{v})\vec{r} + 2(\vec{v}\cdot\vec{r})\vec{v} + 2s\vec{v}\times\vec{r}) = \\
&(0,\ (\cos^2\theta)\vec{r} - (\text{sen}\theta\,\vec{n}\cdot\text{sen}\theta\,\vec{n})\vec{r} + 2(\text{sen}\theta\,\vec{n}\cdot\vec{r})\text{sen}\theta\,\vec{n} + 2\cos\theta(\text{sen}\theta\,\vec{n})\times\vec{r}) = \\
&(0,\ (\cos^2\theta)\vec{r} - (\text{sen}^2\theta\,\vec{n}\cdot\vec{n})\vec{r} + (2\,\text{sen}^2\theta)(\vec{n}\cdot\vec{r})\vec{n} + (2\cos\theta\,\text{sen}\theta)\vec{n}\times\vec{r}) = \\
&(0,\ (\cos^2\theta)\vec{r} - (\text{sen}^2\theta)\vec{r} + (1-\cos 2\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}2\theta)\vec{n}\times\vec{r}) = \\
&(0,\ (\cos^2\theta - \text{sen}^2\theta)\vec{r} + (1-\cos 2\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}2\theta)\vec{n}\times\vec{r}) = \\
&(0, (\cos 2\theta)\vec{r} + (1-\cos 2\theta)(\vec{n}\cdot\vec{r})\vec{n} + (\text{sen}2\theta)\vec{n}\times\vec{r})
\end{aligned} \qquad \textbf{(27)}$$

Observação : na dedução acima, utilizamos as identidades a seguir :

$$\begin{aligned}
&\vec{a}\cdot\vec{a} = |\vec{a}|^2 \\
&\cos^2\theta - \text{sen}^2\theta = \cos 2\theta \\
&2\,\text{sen}^2\theta = (1-\cos 2\theta) \\
&2\cos\theta\,\text{sen}\theta = \text{sen}2\theta
\end{aligned} \qquad \textbf{(28)}$$

Se compararmos agora a parte imaginária da expressão obtida acima com a que derivamos ao final da seção 1.3 para a rotaçao em torno de um eixo, verificamos que são exatamente idênticas com exceção do fator 2 associado ao ângulo θ.

Isso significa que se desejamos aplicar a um ponto $\vec{r}$ uma rotação anti-horária de um ângulo θ ao redor do eixo definido pelo vetor unitário $\vec{n}$, podemos resolver a questão com quatérnios do seguinte modo :

- Represente $\vec{r}$ pelo quatérnio $p = (0, \vec{r})$
- Represente a rotação desejada pelo quatérnio $q = (\cos(\theta/2),\ \text{sen}(\theta/2)\ \vec{n})$

- Realize a operação $R_q(p) = qp\overline{q}$
- A parte real do resultado será zero e a parte imaginária conterá o resultado da rotação

Essa pode parecer, a princípio, uma forma tortuosa de obter o mesmo resultado. Porém, examinemos o que ocorre com a composição de duas rotações. Suponhamos dois quatérnios q_1 e q_2 representando duas rotações distintas. Aplicando sobre um ponto p a rotação composta de q_1 seguida de q_2 obteríamos

$$R_{q2}(R_{q1}(p)) = R_{q2}(q_1 p\overline{q}_1) = q_2 q_1 p\overline{q}_1\overline{q}_2 = q_3 p q_4 \quad \textbf{(29)}$$

Por outro lado, temos que

$$\begin{aligned} q_3 &= q_2 q_1 = (s_2, \vec{v}_2)(s_1, \vec{v}_1) = (s_2 s_1 - \vec{v}_2\vec{v}_1,\ s_2\vec{v}_1 + s_1\vec{v}_2 + \vec{v}_2 \times \vec{v}_1) \Rightarrow \\ \overline{q}_3 &= (s_2 s_1 - \vec{v}_2\vec{v}_1,\ -s_2\vec{v}_1 - s_1\vec{v}_2 - \vec{v}_2 \times \vec{v}_1) \\ &= (s_2 s_1 - \vec{v}_2\vec{v}_1,\ -s_2\vec{v}_1 - s_1\vec{v}_2 + \vec{v}_1 \times \vec{v}_2) \\ q_4 &= \overline{q}_1\overline{q}_2 = (s_1, -\vec{v}_1)(s_2, -\vec{v}_2) = (s_1 s_2 - \vec{v}_1\vec{v}_2,\ -s_1\vec{v}_2 - s_2\vec{v}_1 + \vec{v}_1 \times \vec{v}_2) = \overline{q}_3 \end{aligned} \quad \textbf{(30)}$$

Isto significa que a rotação composta pode ser representada diretamente por $q_3 = q_2 q_1$, já que

$$q_4 = \overline{q}_3 \Rightarrow q_3 p q_4 = q_3 p\overline{q}_3 = R_{q3}(p) \quad \textbf{(31)}$$

Portanto, com essa parametrização, obtemos uma grande vantagem : a composição de rotações é realizada naturalmente pela própria álgebra dos quatérnios. Se dispomos de dois quatérnios unitários que representam duas rotações de ângulos diferentes em torno de eixos distintos, e desejamos encontrar uma representação para a rotação que resulta da composição dessas duas rotações, basta multiplicarmos os dois quatérnios. Como resultado, obteremos automaticamente um novo quatérnio unitário cuja parte imaginária será um vetor na direção e sentido do eixo da rotação resultante e a parte real o cosseno do ângulo de rotação. Desse modo, estaremos sempre descrevendo nossas rotações da forma "natural" (e única salvo o sentido de rotação) que buscávamos.

Para ilustrar esse fato, voltemos ao nosso exemplo da orientação de um avião. Resolvamos, agora com quatérnios, o exemplo em que o piloto, inicialmente voando para o norte, primeiro realiza uma rotação de 90 graus em torno do eixo leste/oeste (voltando o nariz para o solo) e depois outra de 90 graus em torno do eixo sul/norte (voltando a asa esquerda para o céu).

A primeira rotação será representada pelo quatérnio

$$\begin{aligned} q_1 &= (\cos(90/2), \text{sen}(90/2)(0,1,0) \\ &= (\sqrt{2}/2, (\sqrt{2}/2)(0,1,0)) \\ &= (\sqrt{2}/2, (0,(\sqrt{2}/2),0)) \end{aligned} \quad \textbf{(32)}$$

E a segunda rotação pelo quatérnio

$$\begin{aligned} q_2 &= (\cos(90/2), \text{sen}(90/2)(1,0,0) \\ &= (\sqrt{2}/2, (\sqrt{2}/2)(1,0,0)) \\ &= (\sqrt{2}/2, ((\sqrt{2}/2),0,0)) \end{aligned} \quad \textbf{(33)}$$

A rotação composta, portanto, será

$$\begin{aligned} &q_3 = \\ &q_2 q_1 = \\ &(\sqrt{2}/2, ((\sqrt{2}/2), 0, 0))(\sqrt{2}/2, (0, (\sqrt{2}/2), 0)) = \\ &((\sqrt{2}/2)(\sqrt{2}/2) - ((\sqrt{2}/2), 0, 0)(0, (\sqrt{2}/2), 0), (\sqrt{2}/2)(0, (\sqrt{2}/2), \\ &0) + (\sqrt{2}/2)((\sqrt{2}/2), 0, 0) + ((\sqrt{2}/2), 0, 0)\times(0, (\sqrt{2}/2), 0)) = \\ &(½ - (0+0+0), (0, ½, 0) + (½, 0, 0) + ½(1,0,0)\times(0,1,0)) = \\ &(½, (½, ½, 0) + ½(0,0,1)) = \\ &(½, (½, ½, 0) + (0,0, ½)) = \\ &(½, (½, ½, ½)) \end{aligned} \quad \textbf{(34)}$$

Isso corresponde a uma rotação anti-horária de $\theta = 2\arccos(½) = 120$ graus em torno do eixo definido pelo vetor (½, ½, ½).

Para conferir que esta rotação simples em torno de um único eixo realmente corresponde à composição das duas rotações dadas, vamos aplicá-la ao nosso avião. Suponhamos um ponto na asa direita do avião, com ele voando em sua orientação inicial, voltado para o norte. Imaginemos que esse ponto ocupe a posição $\vec{r}_1 = (10,-5,0)$, ou seja, a dez unidades a norte da origem, cinco para o leste e a zero de altura. Já sabemos que após as duas rotações, o avião deverá estar voando para o oeste, com a asa direita apontando para o solo, ou seja, com $\vec{r}_2 = (0,10,-5)$.

Aplicando a rotação deduzida acima sobre $\vec{r}_1$, obtemos :

$q_3(0,\vec{r}_1)\overline{q}_3 =$

$(½, (½, ½, ½))(0, (10,-5,0))(½, –(½, ½, ½)) =$

$(½, (½, ½, ½))(0 + (10,-5,0)\cdot(½, ½, ½), 0 + ½(10,-5,0) – (10,-5,0)\times(½, ½, ½))$
$=$

$(½, (½, ½, ½))((½10 – ½5+ 0), (½10, –½5, 0) – (–½5 – 0, 0 – ½10, ½10 – (–$
$½5))) =$

$(½, (½, ½, ½))(½5, (½10, –½5, 0) – (–½5, –½10, ½15)) =$

$(½, (½, ½, ½))(½5, (½15, ½5, –½15)) =$

$(¼5 – (½, ½, ½)\cdot(½15, ½5, –½15), ½(½15, ½5, –½15) + ½5(½, ½, ½) +$

$(½, ½, ½)\times(½15, ½5, –½15) =$

$(¼5 – (¼15 + ¼5 – ¼15), (¼15, ¼5, –¼15) + (¼5, ¼5, ¼5) +$

$(–¼15 – ¼5, ¼15 – (–¼15), ¼5 – ¼15) =$

$(¼5 – ¼5, (¼20, ¼10, –¼10) + (–¼20, ¼30, –¼10)) =$

$(0, (0, ¼40, –¼20)) =$

$(0, (0, 10, –5))$ **(35)**

Ou seja, simplesmente multiplicando quatérnios, somos capazes de descobrir a parametrização em coordenadas "naturais" da composição de um número arbitrário de rotações e de aplicar essas rotações a pontos dados. Dessa forma, para representar a orientação de uma entidade, precisamos de somente um quatérnio.

Mais do que isso, estamos livres também do problema de gimbal lock. Não existem eixos preferenciais ou perda de graus de liberdade nesta parametrização. A partir de qualquer rotação ou posição, podemos aplicar qualquer outra rotação, sem restrições.

2.3.2 Interpolação de orientações

Podemos agora retornar à questão da interpolação de orientações. Dadas duas orientações, representadas por um quatérnio cada uma, como determinar os quatérnios que representam uma seqüência de orientações intermediárias?

Poderíamos, ingenuamente, imaginar que seria suficiente executar uma interpolação linear entre os dois quatérnios, componente a componente. Esse procedimento, porém, não gera o resultado desejado. É fácil verificar que é esse o caso através de um exemplo.

Voltando ao nosso avião, imaginemos que desejamos iniciar a interpolação na orientação inicial de "voando para o norte com a barriga para o solo", que é atingida através de uma rotação de zero graus em torno de qualquer eixo, e portanto representada pelo quatérnio

$$q_1 = (\cos(0/2), \text{sen}(0/2)\ \vec{n}) = (1, (0,0,0)) \quad \textbf{(36)}$$

e terminar na orientação "voando para o norte com a barriga para o céu", que é atingida através de uma rotação de 180 graus em torno do eixo sul/norte, e portanto representada pelo quatérnio

$$q_2 = (\cos(180/2), \text{sen}(180/2)(1, 0, 0)) = (0, (1, 0, 0)) \quad \textbf{(37)}$$

Para determinar um único quatérnio intermediário por interpolação linear, basta tomar a média dos dois quatérnios, isto é,

$$q_3 = ½(q_1+q_2) = ½(\ (1, (0,0,0)) + (0, (1, 0, 0))\ ) = (½, (½, 0, 0)) \quad \textbf{(38)}$$

É imediato verificar que o quatérnio q_3 não é a orientação que desejamos. Na verdade, o quatérnio encontrado sequer atende ao requisito para representar uma rotação, pois não é unitário. De fato, se calculamos sua norma, verificamos que

$$|q_3|^2 = (½)^2 + ((½)^2 + 0^2 + 0^2) = ¼ + ¼ = ½ \Rightarrow |q_3| = ½\sqrt{2} \quad \textbf{(39)}$$

Qual a causa disso? Examinando a questão com mais cuidado, verificaremos que o subconjunto dos quatérnios que representa rotações, por ser todo formado por quatérnios unitários, define uma hiperesfera (generalização da esfera para quatro dimensões) de raio unitário, assim como o subconjunto dos complexos que representam rotações define um circulo.

Se desejássemos interpolar entre as orientações planas representadas por dois complexos de forma a obter orientações intermediárias, não poderíamos simplesmente executar uma interpolação linear componente a componente, pois estaríamos percorrendo uma reta secante ao círculo unitário que contém ***todos*** os complexos que representam rotações. Para obter o resultado desejado, deveríamos executar a interpolação ao longo do círculo, dividindo o arco intermediário em segmentos de comprimento igual.

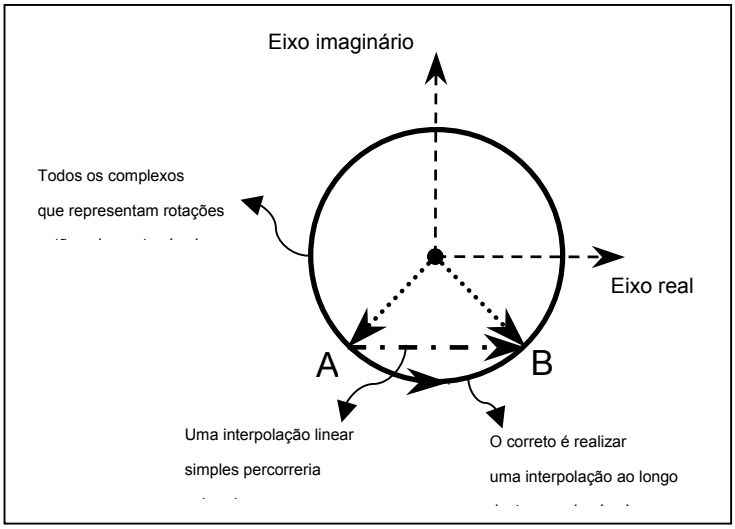

Figura 3 : Interpolação entre orientações com complexos

Com isso, esbarramos em outra questão : ao contrário da interpolação linear, há ***dois*** "caminhos" igualmente válidos para realizarmos a interpolação, pois podemos percorrer o círculo tanto no sentido horário como no anti-horário e em ambos os casos chegaremos à orientação final desejada. A escolha natural é escolher o caminho mais curto (que corresponderá ao menor ângulo de rotação) mas é preciso escolher um dos dois. Mesmo assim, há o caso limite das duas orientações ocuparem posições diamentralmente opostas no círculo e portanto os dois caminhos terem igual comprimento; nessa situação, não há uma escolha "natural" por um ou por outro e somos obrigados a selecionar arbitrariamente um dos dois. Em termos concretos, essa situação surge quando queremos girar uma entidade de 180 graus; podemos girar tanto no sentido anti-horário como no horário que obteremos o mesmo resultado com o mesmo "esforço".

Toda essa discussão se generaliza, sem grandes mudanças, para o caso dos quatérnios e rotações tridimensionais. Ao executar uma interpolação entre dois quatérnios que representam rotações, devemos procurar o caminho mais curto sobre a superfície da hiperesfera unitára que contém todos os quatérnios que representam rotações e dividi-lo em segmentos de comprimento igual.

Executar a operação acima não é tão complicado como possa parecer. Em primeiro lugar, devemos determinar a distância angular entre os dois quatérnios na superfície da hiperesfera. Como ambos são unitários, o produto escalar dos dois produzirá o cosseno do ângulo :

$$q_1 \cdot q_2 = \cos\Omega \Rightarrow \Omega = \arccos(q_1 \cdot q_2) \qquad \textbf{(40)}$$

A expressão para o quatérnio que corresponde a uma posição angular intermediária θ entre q_1 e q_2, ou seja, com um deslocamento entre zero e Ω com relação a q_1, é dada por

$$q_3 = q_1 \frac{\text{sen}(\Omega - \theta)}{\text{sen}\Omega} + q_2 \frac{\text{sen}\theta}{\text{sen}\Omega} \qquad \textbf{(41)}$$

Substituindo θ por $u\Omega$, onde $u \in [0,1]$, obtemos

$$q_3 = q_1 \frac{\text{sen}(1 - u)\Omega}{\text{sen}\Omega} + q_2 \frac{\text{sen}(u\Omega)}{\text{sen}\Omega} \qquad \textbf{(42)}$$

A expressão acima, que damos sem demonstração (embora esta não seja difícil de realizar), produz um quatérnio que está a uma fração u do caminho entre q_1 e q_2.

3 Uma implementação em C das operações com quatérnios

3.1 Descrição da biblioteca

Nesta biblioteca, escrita em C, implementamos rotinas que permitem a realização de todas as operações descritas neste trabalho. O usuário precisa apenas incluir o arquivo cabeçalho "quat.h" e poderá usar qualquer das funções da biblioteca. Foi incluído também um pequeno programa de demonstração que resolve as contas apresentadas ao final da seção 2.4.1.

A biblioteca define e ulitiza três tipos básicos : um para conter quatérnios, dados pelos seus quatro parâmetros usuais, um para conter rotações sob a forma de ângulo em graus e eixo de rotação (descrita no trabalho e utilizada, por exemplo, pelo OpenGL) e, finalmente, um tipo que contém uma posição, em coordenadas cartesianas, no espaço tridimensional.

quatAdd retorna a soma de dois quatérnios.

quatSub retorna a diferença de dois quatérnios.

quatScale retorna o produto de um escalar por um quatérnio.

quatMult retorna o produto de dois quaternios.

quatConj retorna o conjugado de um quatérnio.

quatNorm calcula a norma de um quatérnio.

quatNormalize normaliza um quatérnio (isto é, aplica a ele uma escala tal que ele se torne unitário ou retorna o próprio quatérnio se sua norma for zero).

quatInv calcula o inverso multiplicativo de um quatérnio (isto é, retorna o quatérnio que multiplicado pelo quatérnio dado resulta no valor real 1 ou o próprio quatérnio se sua norma for zero).

quatLinearInterpol retorna um quatérnio intermediário entre dois quatérnios dados, obtido por interpolação linear simples componente a componente e situado a uma fração dada (entre 0 e 1) da distância linear entre os dois. Essa função, apesar de disponibilizada ao usuário, foi implementada basicamente para uso interno pela biblioteca.

quatDot retorna o produto escalar entre dois quatérnios.

quatAngularInterpol retorna um quatérnio intermediário entre dois quatérnios dados, obtido por interpolação angular sobre a hiperesfera unitária e situado a uma fração dada (entre 0 e 1) da distância angular entre os dois.

quatToRotation retorna a rotação codificada por um quatérnio.

quatFromRotation retorna o quatérnio correspondente a uma rotação.

quatToPosition retorna a posição correspondente a um quatérnio.

quatFromPosition retorna o quatérnio correspondente a uma posição.

quatCompose retorna a rotação correspondente à composição de duas rotações sucessivas.

quatRotate retorna a posição correspondente à aplicação de uma rotação dada sobre uma posição dada.

3.2 Listagem da biblioteca

3.2.1 Arquivo quat.h

```
/* quat.h */
/* Copyright (c) 2001 Sergio Biasi <sbiasi@sbiasi.com> */
/* Distribution and use of this is completely free */


#ifndef _QUAT_H_INCLUDED_
#define _QUAT_H_INCLUDED_



typedef struct
{
  double s;
  double vx;
  double vy;
  double vz;
} quat_t;


typedef struct
{
  double theta;
  double nx;
  double ny;
  double nz;
} quat_rot_t;


typedef struct
{
  double x;
  double y;
  double z;
} quat_pos_t;
```

```
quat_t quatAdd(quat_t q1, quat_t q2);
quat_t quatSub(quat_t q1, quat_t q2);
quat_t quatScale(double s, quat_t q1);
quat_t quatMult(quat_t q1, quat_t q2);
quat_t quatConj(quat_t q);
double quatNorm(quat_t q);
quat_t quatNormalize(quat_t q);
quat_t quatInv(quat_t q);
quat_t quatLinearInterpol(quat_t q1, quat_t q2, double t);
double quatDot(quat_t q1, quat_t q2);
quat_t quatAngularInterpol(quat_t q1, quat_t q2, double u);

quat_rot_t quatToRotation(quat_t q);
quat_t quatFromRotation(quat_rot_t r);

quat_pos_t quatToPosition(quat_t q);
quat_t quatFromPosition(quat_pos_t p);

quat_rot_t quatCompose(quat_rot_t r1, quat_rot_t r2);
quat_pos_t quatRotate(quat_pos_t p, quat_rot_t r);


#endif /* _QUAT_H_INCLUDED_ */
```

3.2.2 Arquivo quat.c

```
/* quat.c */
/* Copyright (c) 2001 Sergio Biasi <sbiasi@sbiasi.com> */
/* Distribution and use of this is completely free */

#include <math.h>

#include "quat.h"


const double halfpi = 1.57079632679489661923132169164;
const double epsilon = 0.000001;


/* Sum of two quaternions */
quat_t quatAdd(quat_t q1, quat_t q2)
{
  quat_t qr;

  qr.s = q1.s + q2.s;
  qr.vx = q1.vx + q2.vx;
  qr.vy = q1.vy + q2.vy;
  qr.vz = q1.vz + q2.vz;

  return qr;
}



/* Difference of two quaternions */
quat_t quatSub(quat_t q1, quat_t q2)
{
```

```
  quat_t qr;

  qr.s = q1.s - q2.s;
  qr.vx = q1.vx - q2.vx;
  qr.vy = q1.vy - q2.vy;
  qr.vz = q1.vz - q2.vz;

  return qr;
}



/* Product of a quaternion and a scalar */
quat_t quatScale(double s, quat_t q1)
{
  quat_t qr;

  qr.s = s * q1.s;
  qr.vx = s * q1.vx;
  qr.vy = s * q1.vy;
  qr.vz = s * q1.vz;

  return qr;
}



/* Product of two quaternions */
quat_t quatMult(quat_t q1, quat_t q2)
{
  quat_t qr;

  qr.s = q1.s*q2.s - q1.vx*q2.vx - q1.vy*q2.vy - q1.vz*q2.vz;
  qr.vx = q1.s*q2.vx + q2.s*q1.vx + q1.vy*q2.vz - q1.vz*q2.vy;
  qr.vy = q1.s*q2.vy + q2.s*q1.vy + q1.vz*q2.vx - q1.vx*q2.vz;
  qr.vz = q1.s*q2.vz + q2.s*q1.vz + q1.vx*q2.vy - q1.vy*q2.vx;

  return qr;
}



/* Conjugate of a quaternion */
quat_t quatConj(quat_t q)
{
  quat_t qr;

  qr.s = q.s;
  qr.vx = -q.vx;
  qr.vy = -q.vy;
  qr.vz = -q.vz;

  return qr;
}



/* Norm of a quaternion */
double quatNorm(quat_t q)
{
  return sqrt(q.s*q.s + q.vx*q.vx + q.vy*q.vy + q.vz*q.vz);
}
```

```
/* Normalize a quaternion */
/* (or return the original quaternion if its norm is zero) */
quat_t quatNormalize(quat_t q)
{
  double norm;

  if( (norm = quatNorm(q)) == 0.0)
    return q;

  return quatScale(1.0 / norm, q);
}



/* Multiplicative inverse of a quaternion */
/* (or return the original quaternion if its norm is zero) */
quat_t quatInv(quat_t q)
{
  double norm;

  if( (norm = quatNorm(q)) == 0.0)
    return q;

  return quatScale(1.0 / norm, quatConj(q));
}



/* Linear interpolation of two quaternions */
quat_t quatLinearInterpol(quat_t q1, quat_t q2, double t)
{
  return quatAdd(quatScale((1.0 - t), q1), quatScale(t, q2));
}



/* Dot (scalar) product of two quaternions */
double quatDot(quat_t q1, quat_t q2)
{
  return sqrt(q1.s*q2.s + q1.vx*q2.vx + q1.vy*q2.vy + q1.vz*q2.vz);
}



/* Angular interpolation of two quaternions */
quat_t quatAngularInterpol(quat_t q1, quat_t q2, double u)
{
  double cosomega;
  double sinomega;
  double omega;
  double s1, s2;


  if( quatNorm(quatSub(q1,q2)) > quatNorm(quatAdd(q1,q2)) )
    q2 = quatScale(-1,q2);


  cosomega = quatDot(q1,q2);


  if( (1.0 - cosomega) < epsilon )
```

```
  {
     return quatLinearInterpol(q1, q2, u);
  }


  if( (1.0 + cosomega) < epsilon )
  {
    quat_t q2a;

    q2a.s = -q2.vx;
    q2a.vx = q2.s;
    q2a.vy = -q2.vz;
    q2a.vz = q2.vy;

    s1 = sin( (1.0 - u) * halfpi );
    s2 = sin( u * halfpi );

    return quatAdd(quatScale(s1, q1), quatScale(s2, q2a));
  }


  omega = acos(cosomega);
  sinomega = sin(omega);

  s1 = sin( (1.0 - u) * omega ) / sinomega;
  s2 = sin( u * omega ) / sinomega;

  return quatAdd(quatScale(s1, q1), quatScale(s2, q2));
}



/* Recover a (theta,n) rotation from a quaternion */
/*   theta is the angle of counter-clockwise rotation in degrees */
/*   n is a vector for the axis of rotation */
quat_rot_t quatToRotation(quat_t q)
{
  quat_rot_t rr;

  rr.theta = (acos(q.s)*180)/halfpi;
  rr.nx = q.vx;
  rr.ny = q.vy;
  rr.nz = q.vz;

  return rr;
}



/* Encode a (theta,n) rotation in a unit quaternion */
/*   theta is the angle of counter-clockwise rotation in degrees */
/*   n is a vector for the axis of rotation */
/* (return a quaternion for a null rotation if the vector n is too small) */
quat_t quatFromRotation(quat_rot_t r)
{
  quat_t qr;
  double halftheta;
  double sinhalftheta;
  double axisnorm;

  if( (axisnorm = sqrt(r.nx*r.nx + r.ny*r.ny + r.nz*r.nz)) < epsilon)
  {
    qr.s = 1.0;
```

```
    qr.vx = 0.0;
    qr.vy = 0.0;
    qr.vz = 0.0;

    return qr;
  }

  halftheta = (r.theta*halfpi)/180;
  sinhalftheta = sin(halftheta);
  qr.s = cos(halftheta);
  qr.vx = sinhalftheta * (r.nx/axisnorm);
  qr.vy = sinhalftheta * (r.ny/axisnorm);
  qr.vz = sinhalftheta * (r.nz/axisnorm);

  return qr;
}



/* Recover a position from a quaternion */
quat_pos_t quatToPosition(quat_t q)
{
  quat_pos_t pr;

  pr.x = q.vx;
  pr.y = q.vy;
  pr.z = q.vz;

  return pr;
}



/* Encode a position in a quaternion */
quat_t quatFromPosition(quat_pos_t p)
{
  quat_t qr;

  qr.s = 0.0;
  qr.vx = p.x;
  qr.vy = p.y;
  qr.vz = p.z;

  return qr;
}


/* Compose two rotations using quaternions */
quat_rot_t quatCompose(quat_rot_t r1, quat_rot_t r2)
{
  return quatToRotation(quatMult(quatFromRotation(r2),quatFromRotation(r1)));
}



/* Rotate a given position using a given rotation */
quat_pos_t quatRotate(quat_pos_t p, quat_rot_t r)
{
  return quatToPosition( quatMult( quatFromRotation(r),
                                   quatMult( quatFromPosition(p),
                                             quatConj(quatFromRotation(r)) )
));
}
```

3.2.3 Arquivo quatdemo.c

```
/* quatdemo.c */
/* Copyright (c) 2001 Sergio Biasi <sbiasi@sbiasi.com> */
/* Distribution and use of this is completely free */


#include <stdio.h>

#include "quat.h"


void PrintQuat(quat_t q)
{
  printf("(%01.2f %01.2f %01.2f %01.2f)", q.s, q.vx, q.vy, q.vz);
}


void PrintRot(quat_rot_t rot)
{
  printf("(%01.2f %01.2f %01.2f %01.2f)", rot.theta, rot.nx, rot.ny, rot.nz);
}


void PrintPos(quat_pos_t r)
{
  printf("(%01.2f %01.2f %01.2f)", r.x, r.y, r.z);
}



int main(void)
{
  quat_rot_t rot1;
  quat_rot_t rot2;
  quat_t q1;
  quat_t q2;
  quat_t q3;
  quat_pos_t r1;
  quat_pos_t r2;


  rot1.theta = 90;
  rot1.nx = 0;
  rot1.ny = 1;
  rot1.nz = 0;
  printf("rot1 = ");
  PrintRot(rot1);
  printf("\n");

  rot2.theta = 90;
  rot2.nx = 1;
  rot2.ny = 0;
  rot2.nz = 0;
  printf("rot2 = ");
  PrintRot(rot2);
  printf("\n");

  printf("\n");
```

```
  q1 = quatFromRotation(rot1);
  printf("q1 = ");
  PrintQuat(q1);
  printf("\n");
  printf("rot(q1) = ");
  PrintRot(quatToRotation(q1));
  printf("\n");

  q2 = quatFromRotation(rot2);
  printf("q2 = ");
  PrintQuat(q2);
  printf("\n");
  printf("rot(q2) = ");
  PrintRot(quatToRotation(q2));
  printf("\n");

  q3 = quatMult(q2,q1);
  printf("q3 = q2*q1 = ");
  PrintQuat(q3);
  printf("\n");
  printf("rot(q3) = ");
  PrintRot(quatToRotation(q3));
  printf("\n");

  printf("\n");

  r1.x = 10;
  r1.y = -5;
  r1.z = 0;
  printf("r1 = ");
  PrintPos(r1);
  printf("\n");


  r2 = quatRotate(r1,rot1);
  printf("rot(r1,q1) = ");
  PrintPos(r2);
  printf("\n");

  r2 = quatRotate(r2,rot2);
  printf("rot(rot(r1,q1),q2) = ");
  PrintPos(r2);
  printf("\n");

  r2 = quatRotate(r1, quatToRotation(q3));
  printf("rot(r1,q3) = ");
  PrintPos(r2);
  printf("\n");


  return 1;
}
```