

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Programa Unificado de Bolsas - Edital 2019/2020
Relatório Final de Iniciação Científica

Animação Computacional Baseada em Modelos Físicos

Bolsista: Caroline Santos Corrêa
Orientador: Prof. Dr. Afonso Paiva Neto

Junho de 2020

Sumário

1	Introdução	2
2	Objetivo	2
3	Introdução à Dinâmica do Corpo Rígido	3
3.1	Sistemas de Coordenadas	3
3.2	Movimento Circular	4
3.3	Matrizes de Orientação	6
3.4	Centro de Massa	7
3.5	Momento Linear e Torque	12
3.6	Momento de Inércia	14
3.7	Exemplos de Cálculo de Momento de Inércia	14
3.8	Implementando momento de inércia 2D	17
4	Implementação com quaternions	17
4.1	Introdução	18
4.2	Quaternions	19
4.3	Quaternion Inverso	21
4.4	Quaternions de rotação	21
4.5	Matriz de rotação \times quaternion de rotação	24
4.6	Atualizando o estado de corpos rígidos	35
5	Introduzindo Colisões	36
5.1	Colisões com planos	36
6	Conclusões	43
7	Referências	44

1 Introdução

A Computação Gráfica é a área da computação destinada à geração de imagens em geral — em forma de representação de dados e informação, ou em forma de recriação do mundo real. Ela pode possuir uma infinidade de aplicações para diversas áreas, desde a própria informática, ao produzir interfaces gráficas para software, sistemas operacionais e sites na Internet, quanto para produzir animações e jogos.

É uma área que se faz presente em nosso cotidiano e possui meios de trazer avanços tecnológicos relevantes para a sociedade. Hoje em dia um dos impactos da pesquisa em computação é o aprimoramento de equipamentos médicos utilizados para treinamentos de futuros doutores e doutoras e/ou para diagnósticos clínicos, como softwares que simulam o funcionamento do corpo humano e a fisionomia dos órgãos internos. Outro uso da simulação computacional que causa impacto social positivo é para aquisição de carta de habilitação, que já está sendo utilizada e tem potencial de fornecer ao indivíduo sob treinamento uma melhor experiência com os carros antes das aulas práticas, oferecendo menos risco para o trânsito das cidades. Todas as imagens envolvidas nessas simulações são resultados de estudos em Computação Gráfica.

A computação gráfica também exerce grande impacto sobre o nosso entretenimento e dia-a-dia na sociedade de consumo. Diversos filmes de longa metragem, sejam animações ou live actions, assim como comerciais de smartphones e carros. Áreas novas com grande aplicações da computação gráfica também estão surgindo, como a realidade aumentada, realidade virtual, tecnologias interativas e visualização de dados.

A compreensão das ferramentas técnicas por trás de animações 3D exige a revisão de conceitos de geometria euclidiana, o estudo de algoritmos e estruturas de dados utilizados na resolução de problemas geométricos e os conceitos físicos envolvidos na modelagem destes elementos virtuais.

Neste contexto, é possível propiciar aos alunos a experiência com aplicações de diversos assuntos vistos em disciplinas de graduação em Matemática, Física e Computação. Esta experiência pode ser de grande uso tanto para a opção por uma carreira acadêmica, quanto para sua inserção no mercado de trabalho, uma vez que há demanda de profissionais em ambas as áreas.

O projeto abrangeu esses assuntos e possibilitou o estudo introdutório da área, tanto teórico quanto computacional. Pudemos realizar algumas simulações apresentadas nas referências, utilizando a linguagem Python.

Todavia, não atingimos o final da vigência da bolsa porque a aluna iniciou um estágio e optou por priorizá-lo para não se sobrecarregar com a graduação. As atividades do projeto se encerraram no dia 30 de junho de 2020 e este relatório contém todas as atividades realizadas desde o início da bolsa, em setembro de 2019.

2 Objetivo

O principal objetivo deste projeto de pesquisa é o estudo dos problemas de Geometria Computacional relacionados a animações 3D e à Física de sua construção, tendo em

vista as vastas aplicações desta área de pesquisa em Computação Gráfica e Modelagem Geométrica. Serão estudadas a dinâmica de corpos rígidos criados computacionalmente e seus movimentos simulados com algoritmos embasados na teoria física que descreve esta dinâmica.

Neste relatório, serão explicados os métodos utilizados e toda a teoria na qual foram embasados. A principal referência do projeto foi o livro **Foundations of physically based modeling and animation**, de House, Donald Keyser e John C.

Exporemos os estudos realizados pela aluna, o conteúdo dos seminários apresentados e as simulações computacionais com os respectivos resultados visuais (plot de gráficos e imagens).

3 Introdução à Dinâmica do Corpo Rígido

Vamos estudar as seções 9.1 a 9.3.1 do livro.

3.1 Sistemas de Coordenadas

Sempre que começamos a estudar assuntos que envolvem movimento de objetos, é importante definir em relação a que, ou a quem, queremos que eles se movam. Daí, falar sobre referências e sistemas de coordenadas é essencial.

Existem dois sistemas de coordenadas: global e local. A definição diz respeito ao modo como escolhemos visualizar os objetos que desenhemos.

Por exemplo, imagine um círculo, de raio 5, desenhado sobre um plano cartesiano, e centrado em (4, 4). Em relação às coordenadas globais, que são x e y (sistema cartesiano), o círculo está deslocado 2 unidades à frente e acima da origem (0, 0).

Para as coordenadas locais, podemos escolher qual a origem do sistema, com base no círculo. Poderíamos dizer, por exemplo, que a origem está no centro dele, em (4, 4). Daí, qualquer interação de outros objetos deverá ser pensada para a origem ser em (4, 4), todo movimento é feito com relação ao círculo.

É possível passar de um sistema de coordenadas para outro, através de uma matriz de rotação R . Vamos ver um exemplo de uma matriz assim, no sistema cartesiano mesmo:

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (1)$$

Essa matriz rotaciona vetores coluna em θ radianos, no sentido anti-horário:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \quad (2)$$

Por exemplo, o vetor $\vec{v} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ rotacionado em θ radianos possui novas coordenadas $\begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$.

Em \mathbb{R}^3 , a matriz de rotação que segue o padrão da anterior, ou seja, rotaciona o vetor em θ radianos, no sentido anti-horário, é da seguinte forma:

$$R(\hat{z}, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3)$$

O eixo de rotação \hat{z} e o ângulo θ são passados como argumento para R .

Note que existe um padrão para obtermos os pontos em novas coordenadas. Ainda no exemplo em \mathbb{R}^2 , chame $p = (x', y')$ e $p_0 = (x, y)$. Temos $p = x + R p_0$, expressão que vem diretamente da multiplicação de matriz feita acima.

Esta é a forma geral para qualquer matriz de orientação, ela não precisa ser somente de rotação; pode alterar as coordenadas do objeto de diversas formas: pode rotacionar e refletir, por exemplo.

O que queremos saber agora é como se comporta um objeto em movimento após rotacionar suas coordenadas, ou seja, como é definida p' , a derivada do ponto p , em relação ao tempo?

Com a regra do produto, chegamos ao seguinte:

$$p'(t) = x'(t) + R'(t)p_0(t) + R(t)p'_0(t) \quad (4)$$

O ponto p_0 está nas coordenadas locais. Ele não vai mudar de posição enquanto o objeto estiver se movendo nas coordenadas globais. Sua posição é constante e sua velocidade é zero. Então,

$$p'(t) = x'(t) + R'(t)p_0(t) \quad (5)$$

Sabemos quem é x' : a velocidade do objeto. Agora, como fica a matriz R' ?

3.2 Movimento Circular

Existe uma mudança de orientação envolvida na troca de coordenadas, que é, justamente, o papel dessa matriz. Se queremos que nosso objeto gire, precisamos introduzir conceitos de movimento circular, para estudarmos como se comporta a matriz de orientação durante o deslocamento.

Vamos explicar o caso 2D, primeiro.

Observe a Figura 1, abaixo:

Imagine que P é uma partícula que está girando em torno da origem desta circunferência, de raio $|r|$. No movimento circular, temos dois tipos de velocidade:

1) A velocidade angular ω , que mede a taxa de variação do ângulo θ de rotação ao longo do tempo, ou seja,

$$\omega = \frac{d\theta}{dt} \quad (6)$$

2) A velocidade linear \vec{v} , que nos dá o deslocamento do vetor posição r , ou seja, a velocidade com que se percorre a circunferência (pense no desenho mesmo, a velocidade

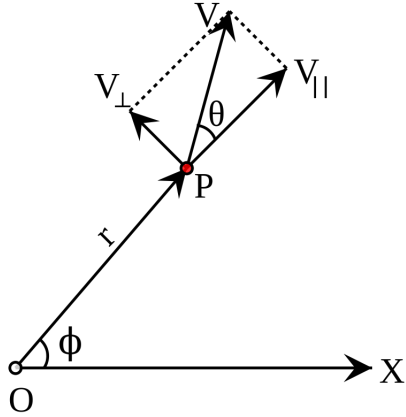


Figura 1: Esquema do movimento circular em duas dimensões

com que "se desenha" o círculo conforme P se move). A este trajeto damos o nome de arco de circunferência l e o obtemos fazendo $l = |r|\theta$. Então,

$$|v| = \frac{dl}{dt} = \frac{d(r\theta)}{dt} = \omega|r| \quad (7)$$

A velocidade linear \vec{v} é vetorial! Ela possui duas componentes: a radial $v_{||}$, ou paralela ao raio da circunferência, e a tangencial v_{\perp} , ou perpendicular ao raio. Com os nomes, podemos identificar essas componentes na figura.

A componente que, de fato, contribui para a variação do ângulo é a tangencial, porque a paralela mantém o ângulo constante. Imagine que não tivéssemos a componente tangencial. A partícula P se deslocaria em linha reta a partir da origem, sem mudar de direção, porque nada diz a ela que gire.

Observando o triângulo formado pelas componentes de \vec{v} , temos $\sin(\theta) = \frac{v_{\perp}}{|v|}$.

Com (7), temos:

$$\omega r = \frac{|v_{\perp}|\sin(\theta)}{r} = \frac{|v|}{|r|} \quad (8)$$

Agora, vamos analisar o caso em \mathbb{R}^3 :

Observe a Figura abaixo, imaginando que o ponto P está girando nessa circunferência, no plano xyz (o objetivo era fazer um desenho com os 3 eixos, mas ainda não descobri como fazer isso no GeoGebra.)

b é o raio da circunferência, ou seja, a distância do ponto P ao centro do eixo de rotação, que é $|r|$. Este valor é constante. Vamos considerar que a velocidade angular não muda, então o raio não varia. O vetor \vec{r} , que forma um ângulo θ com o eixo y , continua sendo o vetor posição de P , cujo deslocamento queremos estudar.

Note que temos $l = b\theta$ para arco de circunferência. Então

$$l = b\theta \implies \frac{dl}{dt} = b\frac{d\theta}{dt} = b\omega \quad (9)$$

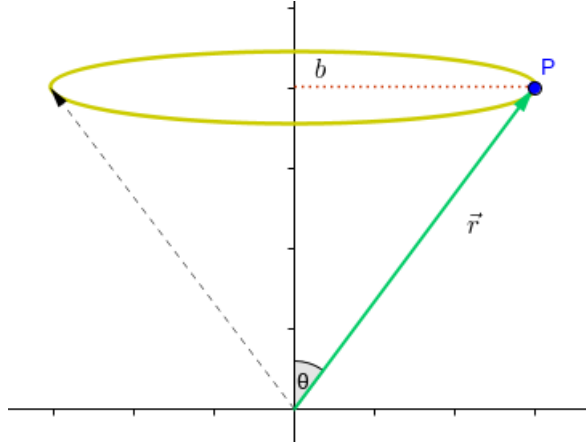


Figura 2: Esquema do movimento circular em três dimensões

Assim, como $\frac{dl}{dt}$ é a velocidade linear \vec{v} , temos $|v| = |\omega||b|$.

Observando a imagem, percebemos que $|b| = |r|\sin(\theta)$. Substituindo na expressão que encontramos para \vec{v} , obtemos que $|v| = |\omega||r|\sin(\theta)$, que é, justamente, a definição do produto vetorial de $\vec{\omega} \times \vec{r}$.

Com a regra da mão direita, obtemos a direção dessa velocidade linear. No caso em que o objeto gire em sentido anti-horário, este vetor apontaria para cima (fora do papel, em direção do teto), conforme nos diz a definição de produto vetorial.

3.3 Matrizes de Orientação

Agora que definimos todos esses pontos, podemos voltar à matriz de orientação R , que era o nosso objetivo. Vamos representá-la dessa forma:

$$R = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix}$$

Cada uma das colunas representa as coordenadas dos pontos do corpo, posicionados nos eixos x, y e z , depois de passarmos para o sistema de coordenadas global.

Por exemplo, considere o vetor $(1, 0, 0)$, que nada mais é do que o eixo x , em coordenadas locais. Num dado instante t , este vetor possui coordenadas $R(t) \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, como

vimos no primeiro exemplo de matriz de rotação.

Se R é tal como definimos acima, temos:

$$\begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad (10)$$

que é justamente a primeira coluna de R , que nos dá a direção do eixo x nas coordenadas globais!

Então, R' é simplesmente a derivada de cada uma das colunas de R . Como essas colunas nos dão a posição do objeto em cada eixo, a derivada deles, isto é, o comportamento deles conforme o objeto se move, nos dará, justamente, a velocidade linear.

Note que como temos 3 eixos e, como vimos, a velocidade angular é sempre por eixo (não tem como girar em torno de mais de um eixo ao mesmo tempo!).

Também teremos uma velocidade linear por eixo, ou seja, uma por coluna da matriz:

$$R' = \left(\omega \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad \omega \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad \omega \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right) \quad (11)$$

Lembre-se de que a velocidade angular $\vec{\omega}$ é constante e representada por $\vec{\omega} = [\omega_x \ \omega_y \ \omega_z]$.

E assim é definido o deslocamento do objeto nas coordenadas globais. Perceba que o custo computacional para se calcular estes 3 produtos vetoriais é alto.

Existe uma alternativa para esse cálculo, dada pela relação $\omega \times r = (\omega*)r$, onde

$$\omega* = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad (12)$$

Dessa forma, simplificamos o cálculo de cada coluna da matriz R' . É fácil verificar que a relação funciona, é somente uma continha :)

3.4 Centro de Massa

Definimos o centro de massa como um ponto de um corpo, ou sistema, onde, hipoteticamente, toda a massa se concentra. É uma maneira de facilitar o estudo do movimento de corpos rígidos porque não precisamos levar em consideração as peculiaridades do formato dos objetos para analisar qualquer movimento que ele faça. Facilita bastante, também, estudar casos de interação entre vários corpos, já que basta observar o comportamento de um único ponto para entender como o sistema está se movendo.

Imagine que tenhamos um corpo dividido em n pontos, ou um sistema com n corpos. O importante é que obtenhamos a contribuição das massas de cada n num dado ponto x do eixo sobre o qual estes sistemas estão. Calculamos o centro de massa com a seguinte expressão:

$$C_M = \frac{\sum_{i=1}^n x_i m_i}{\sum_{i=1}^n m_i} \quad (13)$$

Vamos implementar o algoritmo de cálculo do centro de massa de polígonos. Trabalharemos apenas com o caso 2D, por enquanto

Faremos isso da seguinte forma: dado um polígono P com n vértices, vamos reparti-lo em n triângulos, partindo de um ponto arbitrário (escolhemos a origem). Após determinar o baricentro de cada triângulo e suas áreas, somos capazes de encontrar o centro de massa do polígono.

Código do cálculo do centro de massa de polígonos 2D, em Python:

```
from numpy import *
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import *

#Primeiro, criamos uma função para determinar baricentros de triângulos.
#Recebemos um vetor T com as coordenadas do triângulo cujo baricentro
#pretendemos encontrar. T é uma lista de vetores. Uma vez que se trata
#de um triângulo, teremos 3 vetores com 2 vetores de 2 coordenadas cada:

def baricentro(T):
    B = []

    x = 0
    y = 0

    for v in T:
        x += float(v[0])
        y += float(v[1])

    B.append(x/3)
    B.append(y/3)

    return(B)

#Agora, precisamos de uma função para calcular as áreas destes
#triângulos e faremos isso através do determinante, como explicado
#anteriormente:

def area(T):
    w = []
    Q = []

    for v in T:
        w = []
        w.append(float(v[0]))
        w.append(float(v[1]))
```

```

Q.append(w)

Q = [[z[0],z[1],1.0] for z in Q]
a = np.array(Q)

#print(Q)

return(np.linalg.det(a)/2)

#Uma vez criadas as funções, vamos receber os dados do polígono:

n = input("Número de vértices do polígono:\n")

P = []

for i in range(int(n)):
    v = []
    x,y = input().split(' ')
    v.append(float(x))
    v.append(float(y))
    P.append(v)

#cria os vetores com as coordenadas x e y
Px = [w[0] for w in P]
Py = [w[1] for w in P]

#adiciona o primeiro ponto do polígono ao final do vetor, para fechar
Px.append(Px[0])
Py.append(Py[0])

plt.plot(Px,Py, 'r.-')
plt.title("Polígono")

savefig("x" + str(x) + ".png", dpi=300, bbox_inches='tight')

#Agora, precisamos criar os triângulos que tornarão possível a
#determinação do centro de massa:

TRI = []

for i in range(int(n)):
    v = []

```

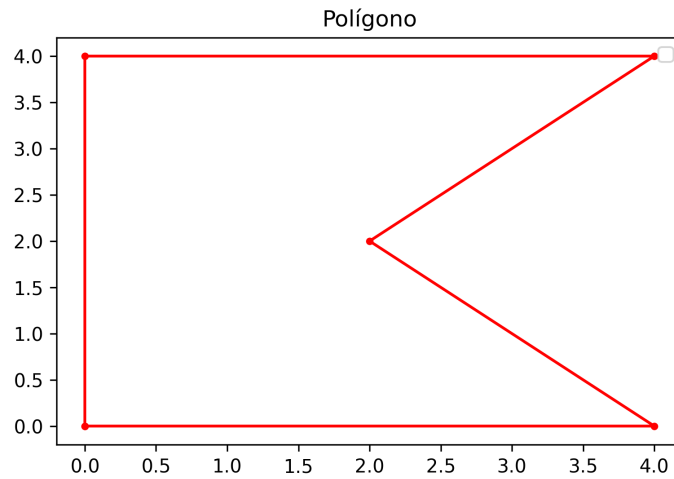


Figura 3

```

v.append([0.0,0.0])
v.append(P[i])
v.append(P[(i+1)%int(n)])
TRI.append(v)

Tx = []
Ty = []

for T in TRI:
    Tx.append([w[0] for w in T])
    Ty.append([w[1] for w in T])

Tx.append([0,0])
Ty.append([0,0])

plt.plot(Tx[0],Ty[0], 'r--')
plt.plot(Tx[1],Ty[1], 'b--')
plt.plot(Tx[2],Ty[2], 'm--')
plt.plot(Tx[3],Ty[3], 'c--')
plt.plot(Tx[4],Ty[4], 'g--')
plt.plot(Tx[5],Ty[5], 'y--')

plt.title("Divisão do polígono em triângulos para o cálculo da área")

savefig("x" + str(x) + ".png", dpi=300, bbox_inches='tight')

#Vamos passar as informações dos triângulos para as funções e

```

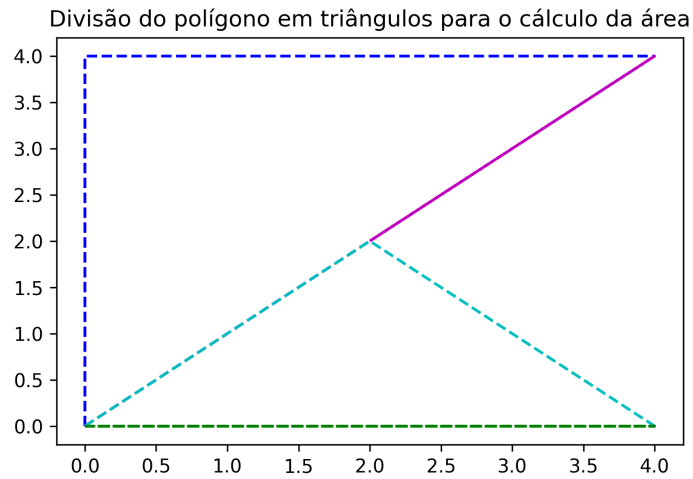


Figura 4

#calcular seus respectivos baricentros e áreas:

```
soma_area = 0
wp = [0,0]
```

```
for i in range(int(n)):
    AREA = area(TRI[i])
    BARICENTRO = baricentro(TRI[i])
```

```
    wp[0] += AREA*BARICENTRO[0]
    wp[1] += AREA*BARICENTRO[1]
```

```
    soma_area += AREA
```

```
print('\n')
```

```
print("Área Total: %.2f" % (soma_area))
```

Output: Área Total:-12.00

#E, finalmente, o centro de massa do polígono:

```
CM = [0,0]
CM[0] = wp[0]/soma_area
CM[1] = wp[1]/soma_area
```

```

print("Centro de massa do polígono: (%.2f,%.2f)" % (CM[0],CM[1]))

plt.plot(Px,Py, 'b.-', label = 'Polígono')
plt.plot(CM[0],CM[1], 'r.',label = 'Centro de Massa')

plt.title("Centro de massa do polígono")

savefig("x" + str(x) + ".png", dpi=300, bbox_inches='tight')

```

Output: Centro de massa do polígono: (1.56,2.00)

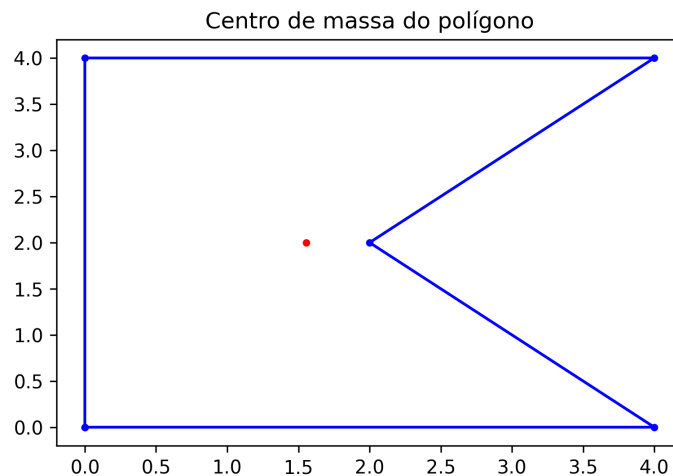


Figura 5

3.5 Momento Linear e Torque

Ainda utilizando a divisão do polígono em triângulos, podemos determinar seu momento de inércia, que é a grandeza responsável por medir o quanto de torque precisamos para fazer um copo rígido girar em torno de um eixo.

Antes de já introduzir momento de inércia e torque, vamos voltar um pouco no conceito de inércia. Ela é uma propriedade de toda matéria e, basicamente, nos diz que um corpo não altera seu estado de movimento, ou seja, sua velocidade, até que seja submetido à uma aceleração.

Então, um objeto que está parado não vai se mover enquanto não dermos um empurrãozinho e, uma vez em movimento, só vai parar, ou alterar sua velocidade, se alguma coisa acontecer: atrito com a superfície em que estiver andando, algo, de fato, o parar, resistência do ar etc. Ou seja, o corpo só altera seu estado quando submetido a alguma força externa.

Existem grandezas físicas responsáveis por medir o quanto de inércia um objeto deve "vencer" para ser colocado em movimento. Nos movimentos lineares, temos o momento

linear, $\vec{\rho}$, definido por:

$$\vec{\rho} = m\vec{v} \quad (14)$$

onde m é a massa do objeto e v sua velocidade. Essa grandeza é vetorial, e medida em kg.m/s.

Se a variação do momento linear não foi nula, num dado intervalo de tempo, dizemos que houve interferência de uma força externa ao sistema que estamos levando em consideração. Daí, dizemos que o momento linear não foi conservado e que a força resultante sobre o sistema não é nula:

$$F_r = \Delta p \quad (15)$$

Se conservado, temos $F_r = \Delta p = 0$.

Se não houve conservação do momento $F_r = \Delta p \neq 0$.

O efeito que vemos num objeto cujo momento linear não foi conservado, uma vez que estamos tratando de uma grandeza vetorial, é mudança na velocidade e/ou na direção do movimento.

Passando para o movimento circular, o raciocínio é o mesmo! Em vez do momento linear, temos o momento angular \vec{L} , que também é uma grandeza vetorial e vai nos dizer o grau de dificuldade para se colocar um corpo para girar em torno de um dado eixo, ou para fazê-lo parar de girar.

Novamente, se o momento angular se mantiver constante ($\Delta L = 0$), dizemos que há conservação do momento angular e qualquer alteração no movimento de rotação é revelada pela alteração no valor do momento angular. Isso só pode acontecer graças à interferência de uma força externa ao sistema, analogamente aos movimentos lineares.

E aqui introduzimos o torque $\vec{\tau}$, que aparece sempre que o sistema apresentar aceleração, ou seja, não temos força resultante nula e nem, portanto, conservação do momento angular.

$$\vec{\tau} = \vec{r} \times \vec{F} \implies |\tau| = r|F|\sin(\theta) \quad (16)$$

onde \vec{F} é a força aplicada sobre o corpo, a uma distância r do centro do eixo de rotação.

Podemos obter o torque, também, derivando o momento angular, uma vez que ele é uma grandeza que mede, justamente, a taxa de variação de \vec{L} :

$$\begin{aligned} \vec{L} = \vec{r} \times \vec{\rho} \implies \frac{dL}{dt} &= \frac{d}{dt}(r \times \rho) \\ &= \frac{dr}{dt} \times \rho + r \times \frac{d\rho}{dt} \\ &= v \times \rho + r \times F_{\text{res}} \end{aligned} \quad (17)$$

A velocidade linear e o momento linear são paralelos (têm a mesma direção). Então o produto vetorial entre eles é zero:

$$\vec{v} \times \vec{\rho} = |v||\rho|\sin(\theta = 0) = 0 \quad (18)$$

Assim, ficamos com

$$\frac{dL}{dt} = \vec{r} \times \vec{F}_{\text{res}} = \tau \quad (19)$$

3.6 Momento de Inércia

Agora, somente a massa do objeto não é suficiente para analisar o movimento, porque a distribuição de massa em torno do eixo de rotação altera significativamente a rotação do corpo. Para isso, temos o momento de inércia I , que cumpre o papel da inércia nos movimentos lineares.

Quando podemos avaliar a rotação de um objeto considerando-o apenas como uma partícula com massa m , a uma distância r do eixo de rotação escolhido, definimos o momento de inércia assim:

$$I = mr^2 \quad \text{e} \quad I = \frac{|L|}{|\omega|} \quad (20)$$

onde m é a massa do objeto distribuída ao longo do eixo de rotação, r é a distância até o centro de rotação, $|\omega|$ é o módulo da velocidade angular e $|L|$ o módulo do momento angular.

Para os corpos rígidos, o cálculo do momento de inércia exige que olhemos para os objetos não como uma única partícula, mas como um conjunto de n pontos infinitesimais, com massa m_i , que estão posicionados a distâncias r_i do centro de rotação, com $i = 1, \dots, n$:

$$I = \sum_{i=1}^n m_i r_i^2 \quad (21)$$

Note que o momento de inércia é maior quando a massa está distribuída em posições mais distantes do centro de rotação, o que justifica o clássico exemplo da patinadora no gelo. Quando ela aproxima os braços para perto de si, ou seja, concentra maior quantidade de massa em torno do centro de rotação, sua velocidade aumenta, porque o momento de inércia também aumenta.

Podemos, ainda, definir o momento de inércia de forma mais precisa, especialmente para situações em que temos objetos com curvas um pouco complicadas, que dificultam a determinação da posição de cada ponto em relação do centro de rotação.

Considere dm a unidade infinitesimal de massa, a uma distância r do centro de rotação:

$$I = \int_{\text{objeto}} r^2 dm \quad (22)$$

3.7 Exemplos de Cálculo de Momento de Inércia

Queremos determinar o momento de inércia de uma barra de espessura desprezível, ou seja, a consideramos unidimensional, e de largura l . Note que só existe uma possibilidade de rotação, já que estamos num caso unidimensional. Será ao longo da largura da barra.

Lembrando que definimos a densidade $\rho = m/v$ (massa por volume), temos $dm = \rho dr$. Note que não utilizamos nem área, nem volume, para a barra, porque estamos lidando com somente uma dimensão. Aqui, temos densidade linear de massa. Precisamos

somente da largura da barra mesmo e vamos reparti-la em pedaços dr , infinitesimais. Substituindo na integral, temos:

$$I = \int_{r=0}^{r=l} r^2 \rho dr = \rho \left(\frac{r^3}{3} \right) \Big|_0^l = \frac{ml^2}{3}$$

E está feito! A unidade é $kg.m^2$.

Podemos estudar, também, os casos 2D.

Daí, vai ser necessário o uso de integrais duplas, porque precisamos dos pontos de massa que estão sob os dois eixos.

Vamos fazer para o caso de um quadrado, de lado 2, com centro na origem. Temos que encontrar o elemento de área dm da integral. Imagine um pedacinho do quadrado, infinitesimal. A área dA desta região é dada por $dxdy$, já que a área de um quadrado é o produto dos lados. Então,

$$\rho = \frac{m}{A} \text{ e } dA = dx \cdot dy \implies dm = \rho dxdy$$

Vamos escolher o centro de massa do quadrado para ser o centro de rotação, ou seja, o ponto (0,0). A distância de qualquer ponto do quadrado até o centro é $r^2 = x^2 + y^2$ (distância de ponto a ponto, como vemos em GA). Substituindo na integral, então:

$$I = \int r^2 dm = \int_0^2 \int_0^2 \rho(x^2 + y^2) dxdy = \rho \frac{32}{3} = \frac{32m}{12} = \frac{8m}{3}$$

E o processo é o mesmo para qualquer figura 2D:

- 1) Determinar o ponto em torno do qual se deseja que o corpo gire.
- 2) Encontrar o elemento de área dm , a partir da densidade, sempre considerando grandezas infinitesimais!
- 3) Determinar a distância r^2 de qualquer ponto da figura ao centro de rotação escolhido.
- 4) Montar a integral dupla e calculá-la.

A implementação nem sempre é simples, porque cada objeto vai exigir o cálculo de uma integral diferente, especialmente no caso 3D, nos quais as integrais são ainda mais complicadas e exigem que lidemos com o momento de inércia como um tensor.

Para muitos objetos, os cálculos já foram feitos e temos o resultado do momento de inércia:

https://en.wikipedia.org/wiki/List_of_moments_of_inertia

Só um último adendo ao momento de inércia. Por que ele é definido como um tensor, ou seja, uma matriz 3×3 somente quando estamos tratando de rotações em 3 dimensões?

Foram dados dois exemplos neste seminário: uma barra unidimensional, girando no plano e uma placa quadrangular, também girando no plano.

O que importa para o momento de inércia são dois fatores: **distribuição de massa** e **eixo de rotação**. Nos dois exemplos dados, os objetos não eram tridimensionais, mas

estavam girando **plano 2D**, que nos dá apenas uma possibilidade de rotação, que é em torno do ponto escolhido.

Se qualquer um dos dois objetos estivesse sendo rotacionado em 3 dimensões, o resultado do momento de inércia seria, de fato, uma matriz, porque existem 9 possibilidades de rotação para ele.

No entanto, ainda que o objeto gire no plano, existem momentos de inércia em I_x e I_y , e o que obtemos é o **momento de inércia de área**, que serve para qualquer polígono disposto no plano xy .

Perceba que, apesar de só haver um jeito de girar no plano, é inevitável que o rotação "pegue os dois eixos". Daí, o momento de inércia é justamente a contribuição de massa em cada ponto do eixo, infinitesimalmente.

Temos o momento de inércia em cada eixo:

$$I_x = \int \int_R y^2 dA \quad \text{e} \quad I_y = \int \int_R x^2 dA \quad (23)$$

E o total, que foi o que calculamos no exemplos:

$$I_{xy} = \int \int_R x^2 + y^2 dA = I_x + I_y \quad (24)$$

Se estivermos em 3 dimensões, obtemos o momento de inércia como uma matriz 3×3 :

$$\begin{pmatrix} I_{xx} & I_{yx} & I_{zx} \\ I_{xy} & I_{yy} & I_{zy} \\ I_{xz} & I_{yz} & I_{zz} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = \begin{pmatrix} L_x \\ L_y \\ L_z \end{pmatrix} \quad (25)$$

Cada entrada é o momento de inércia em torno de uma das possibilidades de eixo, em 3 dimensões.

Este material contém explicações matemáticas bem convincentes sobre o assunto, caso alguém deseje verificar isso tudo:

http://www.lattice.itp.ru/~pbaivid/lecture_notes/Inertia_Tensor_lecture_notes.pdf

Acredito que esse vídeo pode ilustrar um pouco isso:

https://www.youtube.com/watch?v=_M1MgU0CZ78

Temos um triângulo de 2 dimensões girando **no plano xyz**. Daí, ele tem várias possibilidades de rotação.

3.8 Implementando momento de inércia 2D

Para qualquer polígono, com n vértices, obtemos o momento de inércia da seguinte forma¹:

$$\begin{aligned} I_y &= \frac{1}{12} \sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i) (x_i^2 + x_i x_{i+1} + x_{i+1}^2) \\ I_x &= \frac{1}{12} \sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i) (y_i^2 + y_i y_{i+1} + y_{i+1}^2) \\ I_{xy} &= \frac{1}{24} \sum_{i=1}^n (x_i y_{i+1} - x_{i+1} y_i) (x_i y_{i+1} + 2x_i y_i + 2x_{i+1} y_i) \end{aligned} \quad (26)$$

E aqui está a implementação:

Cálculo do Momento de Inércia de Polígonos 2D, em Python:

```
#Vamos obter, também, o momento de inércia desse mesmo polígono.
#Note que vamos girá-lo no plano 2D. Então, o resultado do momento
#de inércia é um escalar, porque só existe um jeito de girar:

soma = 0

#aqui que o momento é calculado, conforme explicado nas notas do
#seminário 01.
for i in range(int(n)-1):
    soma += (Px[i]*Py[i+1]-Px[i+1]*Py[i])*(Px[i]*Py[i+1]+2*Px[i]*Py[i]
    +2*Px[i+1]*Py[i+1]+Px[i+1]*Py[i])

print("Momento de inércia total: %.2f kg.m^2" % (soma/24))
```

Output: Momento de inércia total: 82.00 kg.m2

Note que se os vértices são inseridos em sentido horário, o momento de inércia será negativo e, se forem colocados em sentido anti-horário, será positivo.

4 Implementação com quaternions

Vamos falar das seções 9.3.2 a 9.4 do livro.

¹A demonstração deste método vem do Teorema de Green e pode ser verificada neste link: <https://leancrew.com/all-this/2018/01/greens-theorem-and-section-properties/>

4.1 Introdução

Já estudamos as matrizes de rotação e como obter as coordenadas de um objeto em movimento a partir dessa matriz. Agora, imagine que queiramos recuperar as informações do estado original de um objeto a partir de sua matriz de rotação derivada. Ou, imagine que você mudou a orientação de um objeto duas vezes seguidas e quer obter a orientação dele após uma das mudanças, ou, até mesmo, seu estado original.

Isso exige nada mais que uma integração, já que, como a derivada da posição, num certo intervalo de tempo, resulta na velocidade, e, se integrarmos a velocidade, obtemos a posição.

Como integrar uma matriz de rotação? Para derivar, simplesmente derivamos cada uma das linhas e obtemos a velocidade linear de rotação em torno de cada eixo. Numericamente, podemos determinar a matriz R' a partir da relação de equivalência que definimos no Seminário 01, sem problemas.

Contudo, integrações numéricas em geral envolvem soma de matrizes, como o método de Euler:

$$R^{[n+1]} = R^n + hR'^n \quad (27)$$

onde h é um passo entre uma integração e outra, que escolhemos.

O problema é que a soma de duas matrizes ortogonais nem sempre é ortogonal. Após integrar R , o resultado poderia não ser mais uma matriz de rotação e perderíamos informações sobre o movimento do objeto.

.....

Relembrando GA

Lembre-se do que significa (e da importância) de a matriz de orientação ser ortogonal. Suas linhas, ou colunas, são ortonormais, ou seja, os vetores que compõem as linhas e as colunas de R possuem produto interno nulo (ortogonais) e são unitários. Também podemos dizer que as linhas, ou colunas, formam uma base ortonormal para o espaço vetorial em que estivermos (em geral, acredito que vamos estar somente em \mathbb{R}^n).

Vamos relembrar o significado de um vetor unitário, em \mathbb{R}^2 .

Seja $\vec{v} \in \mathbb{R}^2$. Denotamos por \hat{v} seu vetor unitário o definimos como

$$\hat{v} = \frac{v}{|v|} \quad (28)$$

sendo $|v|$ a norma/módulo de v . Note que \hat{v} também é um vetor de \mathbb{R}^2 ! Se temos $\vec{v} = (2, 3)$, por exemplo, temos

$$\hat{v} = \frac{(2, 3)}{|2^2 + 3^2|} = \left(\frac{2}{\sqrt{13}}, \frac{3}{\sqrt{13}} \right) \quad (29)$$

Então, note que o vetor unitário \hat{v} é paralelo ao vetor v , ou seja, tem a mesma direção que ele. Daí, temos informação da direção de um vetor guardada num vetor de módulo 1.

Vamos retomar o exemplo do Seminário 01:

$$R = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \quad (30)$$

Queríamos saber as coordenadas do vetor $(1, 0, 0)$ após aplicar a matriz de orientação R e vimos que a seguinte multiplicação:

$$\begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad (31)$$

nos dá, justamente, as novas coordenadas de $(1, 0, 0)$.

Agora, o que garante que esse novo vetor que representa $(1, 0, 0)$ não alterou o comprimento/módulo dele? Seria bem ruim se aplicássemos a matriz de rotação num vetor para rotacioná-lo e o resultado fosse um vetor que não preservasse o tamanho, ou o sentido, do original, já que queríamos apenas girar.

É exatamente a ortogonalidade de R que garante que o módulo dos vetores não seja alterado! O vetor (r_{xx}, r_{xy}, r_{xz}) é unitário e contém a nova direção de \vec{v} . Podemos dizer, sem dúvida, que a única informação contida nele é a nova direção do vetor, sem mudar o módulo dele.

Assim, podemos girar (ou alterar para qualquer orientação) um objeto sem mudar seu formato.

.....

4.2 Quaternions

Para resolver esse problema e melhorar o desempenho numérico da atualização do estado de corpos rígidos, introduzimos os quaternions.

.....

Um pouco sobre números complexos

Vamos retomar algumas definições de números complexos, em \mathbb{R}^2 . Seja $z = a + bi$ um número complexo, com $a, b \in \mathbb{R}$. Definimos o módulo de z como

Como \vec{a} , \vec{b} e \vec{z} são vetores, temos $z^2 = a^2 + b^2 \rightarrow |z| = \sqrt{a^2 + b^2}$ e as operações trigonométricas usuais:

$$\text{sen}(\theta) = \frac{|b|}{|z|} \quad \cos(\theta) = \frac{|a|}{|z|} \quad \text{tg}(\theta) = \frac{|a|}{|b|} \quad (32)$$

O ângulo θ é chamado de argumento do número complexo z . Daí, podemos escrever qualquer número complexo da forma $z = |z|\cos(\theta) + i\text{sen}(\theta)$ e chegar na forma polar:

$$z = |z|(\cos(\theta) + i\text{sen}(\theta)) = |z|e^{i\theta} \quad (33)$$

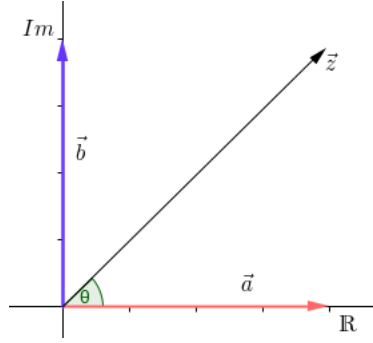


Figura 6: Definindo os números complexos na forma polar

Podemos dizer que os quaternions são vetores de \mathbb{R}^4 dessa forma:

$$q = q_1 + \mathbf{i}q_2 + \mathbf{j}q_3 + \mathbf{k}q_4 \quad (34)$$

sendo $q_1 \in \mathbb{R}$ um escalar que representa a parte real e $q_2, q_3, q_4 \in \mathbb{R}$ escalares que representam a parte imaginária. É conveniente, no nosso contexto, definir $\mathbf{q} = (q_2, q_3, q_4) \in \mathbb{R}^3$ como um vetor.

Vamos começar a utilizar a notação $q = (q_0, \mathbf{q})$, sendo q_0 a parte real e \mathbf{q} a imaginária. Daí, dados dois quaternions q e q^* , definimos o produto como:

$$qq^* = (q_0 + \mathbf{q}(q_0^* + \mathbf{q}^*)) = (q_0q_0^* - \mathbf{q} \cdot \mathbf{q}^*, q_0\mathbf{q}^* + q_0^* \mathbf{q} + \mathbf{q} \times \mathbf{q}^*) \quad (35)$$

Note que a primeira coordenada só possui números reais e a segunda imaginários. O produto de dois quaternions também é um quaternion (preservamos a notação (q_0, \mathbf{q})).

Lembrando que $\mathbf{u} = \frac{\mathbf{q}}{|\mathbf{q}|}$, vamos assumir que só queremos realizar rotações em torno de um mesmo eixo, sempre mudando apenas o ângulo. Então, temos $\mathbf{q} \cdot \mathbf{q}^* = 1$ e $\mathbf{q} \times \mathbf{q}^* = 0$, porque as direções de q e q^* são iguais. Isso que significa não mudamos de eixo a cada rotação, porque, como vimos, os vetores unitários representam a direção do vetor. Falar sobre direção, em rotação, é dizer o eixo em torno do qual estamos girando.

Dessa forma, reduzimos o produto (35) a:

$$q_1q_2 = r_1r_2(\cos(\theta_1 + \theta_2), \hat{u}(\sin(\theta_1 + \theta_2))) \quad (36)$$

sendo r_1 e r_2 os módulos de q_1 e q_2 , isto é:

$$|q_1| = \sqrt{s_1^2 + u_1^2} = \sqrt{a_1^2 + b_1^2 + c_1^2 + d_1^2} \quad |q_2| = \sqrt{s_2^2 + u_2^2} = \sqrt{a_2^2 + b_2^2 + c_2^2 + d_2^2} \quad (37)$$

4.3 Quaternion Inverso

O elemento identidade, ou elemento neutro da multiplicação, dos quaternions é $q = (1, 0)$ (módulo $r = 1$ e argumento $\theta = 0$), pois ¹

$$\begin{aligned} q \cdot (1, 0) &= (s_1, u_1) \cdot (1, 0) \\ &= [r_1(\cos(\theta_1) + \text{sen}(\theta_1))] \cdot [1(\cos(\theta_1 = 0) + \text{sen}(\theta_1 = 0))] \\ &= r_1(\cos(\theta + 0) + \text{sen})(\theta + 0) \\ &= q \end{aligned} \tag{38}$$

De fato, se multiplicarmos qualquer quaternion pelo quaternion $q = (1, 0)$, ele não muda.

Agora, podemos definir o quaternion inverso q^{-1} , que é, por definição de inverso, tal que $qq^{-1} = (1, 0)$. Tome $q^{-1} = (s, -u)$:

$$\begin{aligned} qq^{-1} &= (s, u) \cdot (s, -u) \\ &= [r(\cos(\theta) + \text{sen}(\theta))] \cdot [r(\cos(\theta) - \text{sen}(\theta))] \\ &= r^2(\cos(2\theta) - \text{sen}(2\theta)) \\ &= r^2((2\cos^2(\theta) - 1) - \text{sen}(2\theta)) \\ &= (1, 0) \end{aligned} \tag{39}$$

se $r = 1$ e $\theta = 0$. Definimos implicitamente, para fazer sentido que o elemento identidade seja $(1, 0)$.

Por isso, também, é importante normalizar os quaternions, para que sejam sempre unitários. Essa definição de inverso só funciona se ele for unitário.

4.4 Quaternions de rotação

Como podemos utilizar um quaternion, que é um vetor de \mathbb{R}^4 , para operar um vetor de \mathbb{R}^3 ?

Um quaternion que possui parte real nula é definido como quaternion puro. Essa será a representação de um vetor de \mathbb{R}^3 na forma de um quaternion, ou seja, cada entrada da parte imaginária de \mathbf{q} será uma das coordenadas do vetor que queremos utilizar.

Vamos utilizar a notação $\mathbf{q} = q_1 + q_2 + q_3 + q_4$, daqui em diante para facilitar a visualização de algumas contas.

Sejam $q_0 = q_1$ (parte real) e $\mathbf{q} = (q_2, q_3, q_4)$ (parte imaginária).

Considere o quaternion $q = q_0 + \mathbf{q}$ unitário. Note que $q = q_0^2 + \|\mathbf{q}\|^2 = 1$ e a semelhança com a relação fundamental $\text{sen}^2(\theta) + \cos^2(\theta) = 1$. Então, deve existir um ângulo θ tal que:

$$\begin{aligned} q_0^2 &= \cos^2(\theta) \\ \|\mathbf{q}\|^2 &= \text{sen}^2(\theta) \end{aligned} \tag{40}$$

¹Note que isso significa: (1) fazer a parte real ser 1 e a imaginária 0, já que $\cos(0) = 1$ e $\text{sen}(0) = 0$; (2) normalizar o quaternion para que seja unitário, por isso $r = 1$)

De fato, existe um único θ tal que $q_0 = \cos(\theta)$ e $|\mathbf{q}| = \sin(\theta)$. Basta que $\theta = \arccos(q_0)$ e $\theta = \arcsen(|\mathbf{q}|)$ (as funções \arccos e \arcsen são as inversas de cosseno e seno. Por isso, existe um único θ . Precisamos tomar o intervalo $[0, \pi]$, que é onde seno e cosseno são bijetoras e admitem inversa.)

Agora, definimos o quaternion em função do ângulo θ e do vetor unitário $\mathbf{u} = \mathbf{q}/\|\mathbf{q}\|$:

$$q = \cos(\theta) + \mathbf{u}\sin(\theta) \quad (41)$$

Seja $\mathbf{v} \in \mathbb{R}^3$. Podemos definir um operador linear $L_q : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ tal que

$$\begin{aligned} L_q(\mathbf{v}) &= q\mathbf{v}q^{-1} \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{v} + 2(\mathbf{q} \cdot \mathbf{v})\mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{v}) \end{aligned} \quad (42)$$

é imagem do vetor \mathbf{v} rotacionado em θ em torno do eixo \mathbf{q} .

Note que o operador preserva o módulo de v :

$$q\mathbf{v}q^{-1} = |q| \cdot \|\mathbf{v}\| \cdot |q^{-1}| = |q| \cdot |q^{-1}| \cdot \|\mathbf{v}\| = \|\mathbf{v}\| \quad (43)$$

E preserva a direção de \mathbf{v} em relação a \mathbf{q} . Considere $v = k\mathbf{q}$:

$$\begin{aligned} |L_q(v)| &= q(k\mathbf{q})q^{-1} \\ &= (q_0^2 - \|\mathbf{q}\|^2)(k\mathbf{q}) + 2(\mathbf{q} \cdot (k\mathbf{q}))\mathbf{q} + 2q_0(\mathbf{q} \times (k\mathbf{q})) \\ &= k(q_0^2 + \|\mathbf{q}\|^2)\mathbf{q} \\ &= k\mathbf{q} \end{aligned} \quad (44)$$

Perceba os indícios de que L_q seja um operador de rotação! São as mesmas características que encontramos nas matrizes de orientação.

Dáí, podemos definir, para qualquer quaternion unitário:

$$q = q_0 + \mathbf{q} = \cos(\theta/2) + \mathbf{u}\sin(\theta/2) \quad (45)$$

E, para qualquer vetor $\mathbf{v} \in \mathbb{R}^3$, o operador linear $L_q(\mathbf{v}) = q\mathbf{v}q^{-1}$, que representa uma rotação do vetor \mathbf{v} , em θ radianos, em torno do eixo \mathbf{u} .

Vamos entender de onde vem essa relação do operador linear e por que ele define uma rotação em torno de \mathbf{q} .

Primeiro, considere a decomposição $\mathbf{v} = \mathbf{a} + \mathbf{n}$, sendo \mathbf{a} a componente paralela a \mathbf{q} e \mathbf{n} a perpendicular a \mathbf{q} . Já mostramos que a direção de \mathbf{a} é preservada pelo operador L_q , (44). Vejamos o que ocorre com \mathbf{n} :

$$\begin{aligned} L_q(\mathbf{n}) &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2(\mathbf{q} \cdot \mathbf{n})\mathbf{q} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0(\mathbf{q} \times \mathbf{n}) \\ &= (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0\|\mathbf{q}\|(\mathbf{u} \times \mathbf{n}) \end{aligned} \quad (46)$$

No último passo, lembrando que $\mathbf{u} = \mathbf{q}/\|\mathbf{q}\|$, trocamos \mathbf{q} por $\mathbf{u}\|\mathbf{q}\|$.

Agora, denote $\mathbf{n}_\perp = \mathbf{u} \times \mathbf{n}$. Então, temos:

$$L_q(\mathbf{v}) = (q_0^2 - \|\mathbf{q}\|^2)\mathbf{n} + 2q_0\|\mathbf{q}\|\mathbf{n}_\perp \quad (47)$$

Note que \mathbf{n} e \mathbf{n}_\perp têm o mesmo módulo:

$$\|\mathbf{n}_\perp\| = \|\mathbf{u} \times \mathbf{n}\| = \|\mathbf{n}\| \|\mathbf{u}\| \sin(\pi/2) = \|\mathbf{n}\| \quad (48)$$

porque \mathbf{u} e \mathbf{n} são perpendiculares, por hipótese!

Reescrevendo (42) utilizando (45), temos:

$$\begin{aligned} L_q(\mathbf{n}) &= \left(\cos^2\left(\frac{\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right) \right) \mathbf{n} + \left(2\cos\left(\frac{\theta}{2}\right) \sin\left(\frac{\theta}{2}\right) \right) \mathbf{n}_\perp \\ &= \cos(\theta)\mathbf{n} + \sin(\theta)\mathbf{n}_\perp \end{aligned} \quad (49)$$

São necessárias algumas identidades trigonométricas para chegar a esse resultado:

$$\begin{aligned} \cos(2\theta) &= \cos^2(\theta) - \sin^2(\theta) \implies \cos\left(2\frac{\theta}{2}\right) = \cos(\theta) \\ \sin(2\theta) &= 2\cos(\theta)\sin(\theta) \implies \sin\left(2\frac{\theta}{2}\right) = \sin(\theta) \end{aligned} \quad (50)$$

Perceba que este resultado é a parametrização de uma circunferência centrada na origem e de raio 1, num plano definido por \mathbf{n} e \mathbf{n}_\perp .

Vamos retomar a discussão que fizemos com a Figura 1. Para definir a rotação de um ponto em torno de um determinado eixo, vimos que somente a componente perpendicular ao eixo que de fato contribui para a rotação, enquanto que a paralela permanece inalterada.

Com essa demonstração concluímos que \mathbf{u} é um eixo, porque somente a componente ortogonal a ele apresenta alterações na trajetória e essa alteração consiste em, justamente, percorrer uma circunferência.

Então, o vetor resultante da aplicação do operador linear em \mathbf{n} é uma rotação de θ no plano definido por \mathbf{n} e \mathbf{n}_\perp e mostramos o que queríamos.

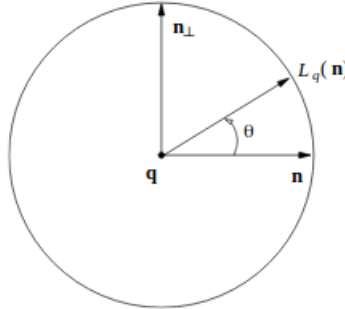


Figura 7: Plano definido pela rotação em torno de \mathbf{q}

Agora, substituindo (44) em (41), obtemos o vetor resultante da aplicação de L_q em um vetor $v \in \mathbb{R}^3$, que é uma rotação de \mathbf{v} , em θ , em torno de \mathbf{q} :

$$\begin{aligned} L_q(\mathbf{v}) &= \left(\cos^2\left(\frac{\theta}{2}\right) - \sin^2\left(\frac{\theta}{2}\right) \right) \mathbf{v} + 2 \left(\mathbf{u} \sin\left(\frac{\theta}{2}\right) \cdot \mathbf{v} \right) \mathbf{u} \sin\left(\frac{\theta}{2}\right) + 2 \cos\left(\frac{\theta}{2}\right) \left(\mathbf{u} \sin\left(\frac{\theta}{2}\right) \times \mathbf{v} \right) \\ &= \cos(\theta) \mathbf{v} + (1 - \cos(\theta))(\mathbf{u} \cdot \mathbf{v}) \mathbf{u} + \sin(\theta)(\mathbf{u} \times \mathbf{v}) \end{aligned} \quad (51)$$

Exemplo

Considere uma rotação em torno do eixo $(1, 1, 1)$, com um ângulo $\frac{2\pi}{3}$. Definimos o vetor unitário:

$$\mathbf{u} = \frac{1}{\sqrt{3}}(1, 1, 1) \quad (52)$$

E encontramos o quaternion que representa a rotação de qualquer quaternion (quaternion puro, ou vetor) em torno de \mathbf{u} , em um ângulo de $\frac{2\pi}{3}$:

$$\begin{aligned} q &= q_0 + \mathbf{q} = \cos\left(\frac{2\pi}{3}\right) + \frac{1}{\sqrt{3}}(1, 1, 1) \sin\left(\frac{2\pi}{3}\right) \\ &= \frac{1}{2} + \frac{1}{2}\mathbf{i} + \frac{1}{2}\mathbf{j} + \frac{1}{2}\mathbf{k} \end{aligned} \quad (53)$$

Agora, imagine que queiramos rotacionar o vetor $\mathbf{i} = (1, 0, 0)$. Utilizamos o operador L_q , como em (49):

$$\begin{aligned} L_q(\mathbf{v}) &= \frac{1}{2}(1, 0, 0) + \left(1 + \frac{1}{2}\right) \frac{1}{\sqrt{3}} \frac{1}{\sqrt{3}}(1, 1, 1) + \frac{\sqrt{3}}{2} \frac{1}{\sqrt{3}}(1, 1, 1 \times (1, 0, 0)) \\ &= \left(-\frac{1}{2}, 0, 0\right) + \left(\frac{1}{2}, \frac{1}{2}, -\frac{1}{2}\right) + \left(0, -\frac{1}{2}, -\frac{1}{2}\right) \\ &= \mathbf{j} \end{aligned} \quad (54)$$

Isso significa que as coordenadas do vetor $(1, 0, 0)$ rotacionado com o quaternion $q = \frac{1}{2} + \frac{1}{2}\mathbf{i} + \frac{1}{2}\mathbf{j} + \frac{1}{2}\mathbf{k}$ são $\mathbf{j} = (0, 1, 0)$ e, nas coordenadas do quaternion original q , são $(0, 1/2, 0)$.

O que conseguimos concluir é que, se queremos rotacionar um vetor \mathbf{v} qualquer em torno de um dado eixo \mathbf{u} , podemos transformar este eixo num quaternion puro e aplicar o operador L_q em \mathbf{v} .

Perceba que rotacionar vetores com quaternions envolve operações mais simples do que com matrizes e não temos os problemas com a ortonormalidade.

4.5 Matriz de rotação \times quaternion de rotação

Podemos definir uma matriz de rotação 3×3 a partir de um quaternion de rotação $q = \cos(\theta/2) + \mathbf{u} \sin(\theta/2)$, porque são estruturas equivalentes.

Devemos ter $q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$ para que q seja unitário, como vimos anteriormente. Caso contrário, não conseguimos obter um quaternion de rotação! Se ocorrer de q não ser unitário, basta normalizá-lo, ou seja, fazer:

$$\mathbf{u} = \frac{\mathbf{q}}{|\mathbf{q}|} = \frac{\mathbf{q}}{\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}} \quad (55)$$

Daí, temos:

$$R(\mathbf{q}) = \begin{bmatrix} q_1^2 + q_2^2 - q_3^2 - q_4^2 & 2(q_2q_3 - q_1q_4) & 2(q_2q_4 + q_1q_3) \\ 2(q_2q_3 + q_1q_4) & q_1^2 - q_2^2 + q_3^2 - q_4^2 & 2(q_2q_4 - q_1q_2) \\ 2(q_2q_4 - q_1q_3) & 2(q_3q_4 + q_1q_2) & q_1^2 - q_2^2 - q_3^2 + q_4^2 \end{bmatrix} \quad (56)$$

Perceba que essa matriz deve ser ortonormal!

Também podemos transformar uma matriz de rotação em um quaternion. O livro não diz qual a ideia por trás do algoritmo. Busquei outras fontes[5].

Se temos uma matriz de rotação, perceba que, ao resolver o sistema de equações a partir das matrizes (56) e

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad (57)$$

a solução nos dá os elementos q_1, q_2, q_3 e q_4 .

Então, equacionando, temos:

$$\begin{aligned} 4q_1^2 &= 1 + r_{11} + r_{22} + r_{33} \\ 4q_2^2 &= 1 + r_{11} - r_{22} - r_{33} \\ 4q_3^2 &= 1 - r_{11} + r_{22} - r_{33} \\ 4q_4^2 &= 1 - r_{11} - r_{22} + r_{33} \\ 4q_3q_4 &= r_{23} + r_{32} \\ 4q_2q_4 &= r_{31} + r_{13} \\ 4q_2q_3 &= r_{12} + r_{21} \\ 4q_1q_2 &= r_{32} - r_{23} \\ 4q_1q_3 &= r_{13} - r_{31} \\ 4q_1q_4 &= r_{21} - r_{12} \end{aligned} \quad (58)$$

Daí, existem 4 possíveis soluções para q_1, q_2, q_3, q_4 . Podemos perceber isso observando as substituições. Encontre as soluções com as 4 primeiras equações e vá substituindo nas seguintes. O processo te dará 4 formas diferentes de escrever q_1, q_2, q_3, q_4 em função das entradas da matriz de orientação.

As soluções são:

$$s_1 = 1/2 \begin{bmatrix} (1 + r_{11} + r_{22} + r_{33})^{1/2} \\ (r_{32} - r_{23})/(1 + r_{11} + r_{22} + r_{33})^{1/2} \\ (r_{13} - r_{31})/(1 + r_{11} + r_{22} + r_{33})^{1/2} \\ (r_{21} - r_{12})/(1 + r_{11} + r_{22} + r_{33})^{1/2} \end{bmatrix} \quad (59)$$

$$s_2 = 1/2 \begin{bmatrix} (r_{32} - r_{23})/(1 + r_{11} - r_{22} - r_{33})^{1/2} \\ (1 + r_{11} - r_{22} - r_{33})^{1/2} \\ (r_{12} + r_{21})/(1 + r_{11} - r_{22} - r_{33})^{1/2} \\ (r_{31} - r_{13})/(1 + r_{11} - r_{22} - r_{33})^{1/2} \end{bmatrix} \quad (60)$$

$$s_3 = 1/2 \begin{bmatrix} (r_{13} - r_{31})/(1 - r_{11} - r_{22} - r_{33})^{1/2} \\ (r_{12} + r_{21})/(1 - r_{11} - r_{22} - r_{33})^{1/2} \\ (1 - r_{11} + r_{22} - r_{33})^{1/2} \\ (r_{23} + r_{32})/(1 - r_{11} + r_{22} - r_{33})^{1/2} \end{bmatrix} \quad (61)$$

$$s_4 = 1/2 \begin{bmatrix} (r_{21} - r_{12})/(1 - r_{11} - r_{22} + r_{33})^{1/2} \\ (r_{31} + r_{13})/(1 - r_{11} - r_{22} + r_{33})^{1/2} \\ (r_{32} + r_{23})/(1 - r_{11} - r_{22} + r_{33})^{1/2} \\ (1 - r_{11} - r_{22} + r_{33})^{1/2} \end{bmatrix} \quad (62)$$

Note que existe uma raiz quadrada em todos os termos das 4 soluções. Dentro das raízes, temos (traço+1), pois o traço de uma matriz é a soma dos elementos da diagonal.

Precisamos garantir que este termo nunca seja negativo, porque queremos resultados reais.

Por isso, é necessário fazer todos aqueles casos do algoritmo:

- Caso 1 do algoritmo e solução s_1 : note que se o traço da matriz for maior ou igual a zero, o termo $(1 - r_{11} - r_{22} + r_{33})$ nunca é negativo. Esta é a solução para ser usada em matrizes com essa propriedade.
- Caso 2 do algoritmo e solução s_2 : temos que o primeiro elemento da diagonal é maior que os outros dois. Daí, o termo $(1 + r_{11} - r_{22} - r_{33})$ é sempre positivo.
- Caso 3 do algoritmo e solução s_3 : aqui, o segundo elemento da diagonal é maior que os outros dois. Então, se fizermos $(1 - r_{11} + r_{22} - r_{33})$, teremos um termo positivo.
- Caso 4 do algoritmo e solução s_4 : por último, temos uma solução para quando o último elemento da diagonal é maior que os outros dois. Daí, basta fazer $(1 - r_{11} - r_{22} + r_{33})$.

Assim, o algoritmo avalia o traço da matriz e utiliza a solução do sistema que melhor se adequa a ela.

Transformando uma matriz de rotação em um quaternion de rotação, em Python:

```
#Seguindo o algoritmo do livro, vamos transformar uma matriz de
#rotação em um quaternion de rotação, utilizando a seguinte
#função:
```

```
from numpy import *
import numpy as np
```

```

from matplotlib.pyplot import *
import math

def matrixtoquat(M,q):

    t = M[0][0]+M[1][1]+M[2][2]
    t = float(t)

    if(t >= 0):
        r = sqrt(t+1)
        q.append(r/2)
        q.append((M[2][1]-M[1][2])/(2*r))
        q.append((M[0][2]-M[2][0])/(2*r))
        q.append((M[1][0]-M[0][1])/(2*r))

    elif((M[0][0] > M[1][1]) and (M[0][0] > M[2][2])):
        r = sqrt(M[0][0]-(M[1][1]+M[2][2]+1))
        q.append((M[2][1]-M[1][2])/(2*r))
        q.append(r/2)
        q.append((M[0][1]-M[1][0])/(2*r))
        q.append((M[2][0]-M[1][2])/(2*r))

    elif((M[1][1] > M[0][0]) and (M[1][1] > M[2][2])):
        r = sqrt(M[1][1]-(M[2][2]+M[0][0]+1))
        q.append((M[0][0]-M[2][0])/(2*r))
        q.append((M[0][1]-M[1][0])/(2*r))
        q.append((M[1][2]-M[2][1])/(2*r))
        q.append(r/2)

    else:
        r = sqrt(M[2][2]-(M[0][0]+M[1][1]+1))
        q.append((M[1][0]-M[0][1])/(2*r))
        q.append((M[2][0]-M[0][2])/(2*r))
        q.append((M[1][2]-M[2][1])/(2*r))
        q.append(r/2)

    return(q)

#Agora, recebemos a matriz, chamamos a função e obtemos q.

M = []
q = []

```

```

print("Digite a matriz, linha por linha:\n")

for i in range(3):
    C = []
    x,y,z = input().split(' ')
    C.append(float(x))
    C.append(float(y))
    C.append(float(z))
    M.append(C)

matrixtoquat(M,q)

print("O quaternion de rotação é:")
print("Parte real: %.2f" % (q[0]))
print("Parte imaginária: (%.2f,%.2f,%.2f)" % (q[1],q[2],q[3]))

#Agora que já sabemos lidar com os quaternions, vamos, finalmente,
#determinar a matriz $$$ de atualização do estado de corpos
#rígidos:

#funções:

def prod_porescalar(A,x):
    C = [[0 for col in range(3)] for row in range(4)]

    for i in range(4):
        for j in range(3):
            C[i][j] = x*A[i][j]

    return(C)

def soma_matriz(A,B):
    C = [[0 for col in range(3)] for row in range(4)]

    for i in range(4):
        for j in range(3):
            C[i][j] = A[i][j]+B[i][j]

    return(C)

def transposta(M):
    X = [[0 for col in range(3)] for row in range(3)]
    for i in range(3):
        for j in range(3):

```

```

X[i][j] = M[j][i]

return(X)

def prodmatrizvet(A,x):
p = []

for i in range(3):
y = 0
for j in range(3):
y += A[i][j]*x[j]
p.append(y)

return(p)

def normaliza(v):
v_unitario = []

modulo = math.sqrt(v[0]**2+v[1]**2+v[2]**2)

v_unitario.append((1/modulo)*v[0])
v_unitario.append((1/modulo)*v[1])
v_unitario.append((1/modulo)*v[2])

return(v_unitario)

def multiplica_quat(q_1,q_2,arg_1,arg_2):
q_1vetor = [q_1[1],q_1[2],q_1[3]]
q_2vetor = [q_2[1],q_2[2],q_2[3]]

r_1 = math.sqrt(q_1[0]**2+q_1[1]**2+q_1[2]**2+q_1[3]**2)
r_2 = math.sqrt(q_2[0]**2+q_2[1]**2+q_2[2]**2+q_2[3]**2)

r_1 = float(r_1)
r_2 = float(r_2)

u = [] #os dois quaternions têm a mesma direção, então têm o mesmo vet.unitário

u.append((1/r_1)*q_1vetor[0])
u.append((1/r_1)*q_1vetor[1])
u.append((1/r_1)*q_1vetor[2])

q_prod = []

```

```

q_prod.append(r_1*r_2*math.cos(arg_1+arg_2))

q_prod.append(math.sin(arg_1+arg_2)*u[0])
q_prod.append(math.sin(arg_1+arg_2)*u[1])
q_prod.append(math.sin(arg_1+arg_2)*u[2])

return(q_prod)

def multiplica_matriz(A,B):

C = [[0 for col in range(3)] for row in range(3)]

for i in range(3):
for j in range(3):
for k in range(3):
C[i][j] += A[i][k]*B[k][j]

return(C)

def quattomatrix(q):

R = [[0 for col in range(3)] for row in range(3)]

R[0][0] = q[0]**2+q[1]**2-q[2]**2-q[3]**2
R[0][1] = 2*(q[1]*q[2]-q[0]*q[3])
R[0][2] = 2*(q[1]*q[3]+q[0]*q[2])

R[1][0] = 2*(q[1]*q[2]+q[0]*q[3])
R[1][1] = q[0]**2-q[1]**2+q[2]**2-q[3]**2
R[1][2] = 2*(q[1]*q[3]-q[0]*q[1])

R[2][0] = 2*(q[1]*q[3]-q[0]*q[2])
R[2][1] = 2*(q[2]*q[3]+q[0]*q[1])
R[2][2] = q[0]**2-q[1]**2-q[2]**2+q[3]**2

return(R)

***** FUNÇÃO PRINCIPAL COMEÇA*****

def ComputeRigidDerivative(S,m,I_0):
dS = []

#obtendo o vetor velocidade:

```

```

p_new = []
p_new.append(S[2][0]/m)
p_new.append(S[2][0]/m)
p_new.append(S[2][0]/m)

dS.append(p_new)

#obtendo a matriz inversa do momento de inércia nas coordenadas globais:

print("Quaternion de rotação:")

q = []
q_x,q_y,q_z,q_k = input().split()
q.append(float(q_x))
q.append(float(q_y))
q.append(float(q_z))
q.append(float(q_k))

#transformando um quaternion numa matriz de orientação:

R = quattomatrix(q)
R_transp = transposta(R)

I = multiplica_matriz(R,I_0)
I_inv = multiplica_matriz(I,R_transp)

for i in range(3):
    for j in range(3):
        I[i][j] = float(I[i][j])

#obtendo a velocidade angular:
omega = prodmatrizvet(I_inv,S[3])

#transformando o vetor omega em um quaternion:

omega_q = [0,omega[0],omega[1],omega[2]]

#obtendo a derivada do quaternion:

print("Argumento do quaternion de rotação:")
arg_q = input()
arg_q = float(arg_q)
print("Argumento do quaternion de omega:")
arg_omega = input()

```



```

arg_omega = float(arg_omega)

dq = multiplica_quat(omega_q,q,arg_q,arg_omega)

dq[1] = 0.5*dq[1]
dq[2] = 0.5*dq[2]
dq[3] = 0.5*dq[3]

dq_puro = [dq[1],dq[2],dq[3]]

dS.append(dq_puro)

#criando o vetor da força resultante(derivada de p):

F = []
F.append(float(0))
F.append(float(0))
F.append(float(0))

dl = []
dl.append(float(0))
dl.append(float(0))
dl.append(float(0))

print("Número de forças aplicadas sobre o corpo(forças externas):")
n = input()
n = int(n)

F = [0,0,0]
for i in range(n):
    print("Vetor força %d:" % i)
    f_x,f_y,f_z = input().split(' ')
    f_x = float(f_x)
    f_y = float(f_y)
    f_z = float(f_z)

    F[0] += f_x
    F[1] += f_y
    F[2] += f_z

#calculando o torque gerado por cada força externa a partir do ponto de
#aplicação
print("Ponto de aplicação da força:")
pt_x,pt_y,pt_z = input().split(' ')

```

```

pt_x = float(pt_x)
pt_y = float(pt_y)
pt_z = float(pt_z)
r = []
r.append(pt_x-x[0])
r.append(pt_y-x[1])
r.append(pt_z-x[2])

dl[0] += np.cross(r,F)[0]
dl[1] += np.cross(r,F)[1]
dl[2] += np.cross(r,F)[2]

dS.append(F)
dS.append(dl)

return(dS)

##### FUNÇÃO PRINCIPAL TERMINA #####

#recebendo dados do corpo:

print("Massa do corpo:")
m = input()
m = float(m)

print("Matriz inversa do momento de inércia:")

I_0 = [[0 for col in range(3)] for row in range(3)]

for i in range(3):
I_0[i][0],I_0[i][1],I_0[i][2] = input().split(' ')

for i in range(3):
for j in range(3):
I_0[i][j] = float(I_0[i][j])

print("Quaternion de rotação:")

q = []
q_x,q_y,q_z,q_k = input().split()
q.append(float(q_x))
q.append(float(q_y))
q.append(float(q_z))
q.append(float(q_k))

```

```

#transformando um quaternion numa matriz de orientação:

R = quattomatrix(q)
for i in range(3):
    for j in range(3):
        R[i][j] = float(R[i][j])

#criando a matriz do estado original do corpo

S = []

print("Vetor posição:")
x=[]
x_x,x_y,x_z = input().split(' ')
x.append(float(x_x))
x.append(float(x_y))
x.append(float(x_z))
S.append(x)

q_puro = [0,q[1],q[2],q[3]]
q_vetor = [q[1],q[2],q[3]]

S.append(q_vetor)

print("Momento linear:")
p = []
p_x,p_y,p_z = input().split(' ')
p.append(float(p_x))
p.append(float(p_y))
p.append(float(p_z))
S.append(p)

print("Momento angular:")
l = []
l_x,l_y,l_z = input().split(' ')
l.append(float(l_x))
l.append(float(l_y))
l.append(float(l_z))
S.append(l)

#criando a matriz derivada do estado do corpo:

print("Matriz do estado original do corpo:")

```

```

for i in range(4):
    print(S[i])

dS = ComputeRigidDerivative(S,m,I_0)
print("Matriz derivada do estado do corpo:")
for i in range(4):
    print(dS[i])

```

4.6 Atualizando o estado de corpos rígidos

Agora que sabemos como evitar o uso da matriz de rotação sempre que quisermos atualizar a posição de um objeto, podemos definir a derivada dos 4 estados dele de forma mais eficiente:

$$S = \begin{pmatrix} \vec{x} \\ R \\ \vec{\rho} \\ \vec{L} \end{pmatrix} \implies S' = \begin{pmatrix} \vec{v} \\ \vec{\omega} * R \\ \vec{F} \\ \vec{\tau} \end{pmatrix} \quad (63)$$

Conforme tudo o que estudamos sobre as grandezas fundamentais que determinam o movimento dos corpos, os elementos em S' podem ser obtidos diretamente de S e, após o estudo sobre os quaternions, podemos substituir a matriz de orientação R pelo quaternion de rotação q . Vamos ver como derivar o quaternion para fazer essa troca:

Para começar, assuma que $q(0) = q_0$ e que $q(1) = q_\omega q_0$, sendo q_ω a velocidade angular escrita como um quaternion ($\omega = (\omega_x, \omega_y, \omega_z) \implies q_\omega = (0, \omega_x, \omega_y, \omega_z)$). Note que, genericamente, isso pode ser escrito como

$$q(t) = q_\omega^t q_0 \quad (64)$$

Agora, lembrando que todo quaternion pode ser representado na forma polar, temos:

$$q_\omega = \cos\left(\frac{\theta}{2}\right) + \sin\left(\frac{\theta}{2}\right) (\mathbf{n}_x i + \mathbf{n}_y j + \mathbf{n}_z k) \quad (65)$$

Lembrando que o termo $(\mathbf{n}_x i + \mathbf{n}_y j + \mathbf{n}_z k)$ representa o eixo em torno do qual estamos girando.

Recordando, também, a forma polar dos números complexos: $z = \exp(it) = \cos(t) + i\sin(t)$, podemos fazer o seguinte:

$$q_\omega = \exp\left(\frac{\theta}{2}\mathbf{n}\right) \implies q_\omega^t = \exp\left(t\frac{\theta}{2}\mathbf{n}\right) \quad (66)$$

Derivando (66) em relação a t , temos:

$$q'(t) = \frac{d}{dt} \left(\exp\left(t\frac{\theta}{2}\mathbf{n}\right) q_0 \right) = \frac{\theta}{2} \mathbf{n} q(t) \quad (67)$$

Como $\theta \mathbf{n}$ é a própria velocidade angular, ficamos com:

$$q'(t) = \frac{1}{2}\omega q(t) \quad (68)$$

Finalmente, podemos montar a matriz S :

$$S = \begin{pmatrix} \vec{x} \\ R \\ \vec{\rho} \\ \vec{L} \end{pmatrix} \Rightarrow S' = \begin{pmatrix} \vec{v} \\ \frac{1}{2}\vec{\omega}q \\ \vec{F} \\ \vec{\tau} \end{pmatrix} \quad (69)$$

Lembrando que precisamos utilizar a fórmula de multiplicação de quaternions (35) para calcular $q'(t)$.

Note como os quaternions são importantes! Se não houvesse uma maneira de substituir a matriz de orientação por alguma outra estrutura, teríamos problemas com a integração.

5 Introduzindo Colisões

O próximo tópico abordado pelo livro são as colisões. Contudo, apesar de o capítulo seguinte falar sobre isso, faz-se, constantemente, menção ao capítulo 3 e à seção 8.5 do livro. Então, será necessário passar por eles antes de ir para o capítulo 10.

Nestas seções o livro analisa o movimento de uma bola 3D colidindo com diversas superfícies e com outros objetos de formatos distintos.

5.1 Colisões com planos

Várias forças estão envolvidas na colisão de um objeto com outro, ou com uma superfície. A análise da colisão envolve estudar um movimento que acontece muito rápido, quase que instantaneamente. Os corpos se deformam por um período de tempo curtíssimo e é esse instante de tempo que queremos estudar aqui. Entender as forças que regem uma colisão e o que é necessário para simular uma, sendo o mais fiel à realidade possível.

Vamos analisar o exemplo de uma bola caindo no chão. Por enquanto, assumiremos que a bola não rotaciona, isto é, não gira em torno do próprio eixo. Dessa forma, nossa bola é, basicamente, um ponto com um raio.

Dividiremos o estudo da colisão em 3 etapas:

Detecção: Ocorreu uma colisão ou não?

Este é, claro, o primeiro passo para começarmos a analisar uma colisão. Afinal, se ela não aconteceu, os outros passos tornam-se irrelevantes.

Observe esta imagem de uma bola de golfe sendo lançada pelo taco, como na Figura 5 abaixo:

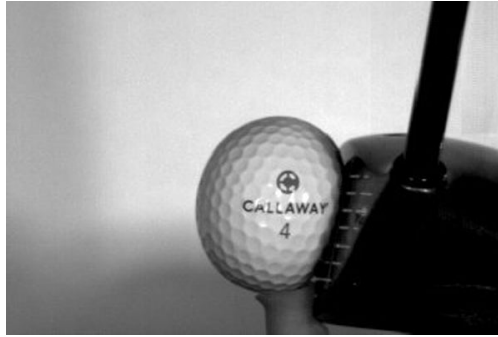


Figura 8: Bola de golfe colidindo no taco

Note que a bola é deformada quando atinge o taco de golfe (vamos ignorar, por hora, fatores como material da bola e do taco que permite que ela deforme mais que o taco, pelo menos conforme a imagem).

Utilizando um esquema de sinais, é possível representar o fato de a bola ter, ou não, colidido olhando para a posição do centro da bola com relação à posição da barreira. Apesar de estarmos lidando com a bola como se fosse um ponto, vamos considerá-la "um ponto com raio", porque precisamos do seu centro como referencial aqui.

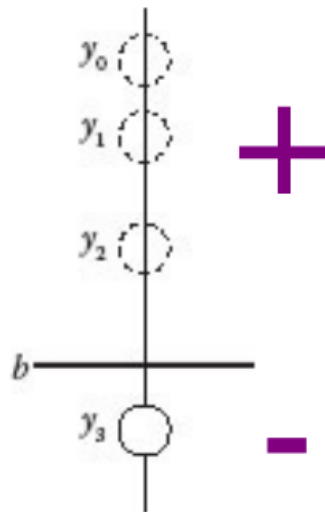


Figura 9: Representação do esquema de sinais para detecção de uma colisão

Assim, se o sinal de $y - b$ for positivo, consideramos que a bola não chegou perto da barreira, e, se for negativo, houve colisão.

Determinação: Onde e em que instante de tempo do intervalo do movimento dos corpos ela aconteceu?

Uma vez que sabemos que a colisão ocorreu, precisamos saber onde no espaço/plano e quando foi.

Por meio da interação da Euler, sabendo a posição do objeto antes e depois da colisão, conseguimos determinar o ponto de colisão.

Considere $d^{[n]}$ a distância no início do movimento e $d^{[n+1]}$ a do final e assuma que a velocidade do corpo foi constante. Daí, obtemos o instante de colisão assim:

$$f = \frac{d^{[n]}}{d^{[n]} - d^{[n+1]}} \quad (70)$$

Retomando o exemplo da bola colidindo com uma barreira, imagine que começamos com 3 unidades à frente do plano e terminamos com 6, isto é, $d^{[n+1]} = -6$ e $d^{[n]} = 3$:

$$f = \frac{d^{[n]}}{d^{[n]} - d^{[n+1]}} = \frac{3}{3 - (-6)} = \frac{1}{3} \quad (71)$$

Ou seja, se o período todo do movimento do objeto foi T , a colisão ocorreu no instante $T/3$.

Daí, para saber a posição em que o objeto estava nesse instante de tempo, escrevendo a equação horária do movimento da bola e aplicando nesse instante, temos a posição exata onde a colisão ocorreu.

Após confirmarmos que a colisão aconteceu e sabermos quando e onde, temos um código para simular a colisão, em que o objetivo é fazer as integrações de Euler para encontrar as posições e instantes de tempo e atualizar o estado dos corpos após a colisão.

Para compreender esse código, precisamos retomar conceitos dados no capítulo anterior (seções de 2.1 a 2.6), que é a origem desse exemplo de uma bola caindo.

Existem algumas equações diferenciais que descrevem o movimento uniformemente variado, isto é, movimentos com aceleração constante e diferente de zero, que são as equações horárias da posição e da velocidade:

$$\text{Posição: } x(t) = x_0 + v_0 t + a \frac{t^2}{2} \quad (72)$$

$$\text{Velocidade: } v(t) = v_0 + at \quad (73)$$

Caso estejamos num caso com velocidade constante (aceleração igual a zero), basta zerar a nessas equações. Lembrando que:

$$a = \frac{dv(t)}{dt} \quad \text{e} \quad v(t) = \frac{dx(t)}{dt} \quad (74)$$

Bem como,

$$v(t) = \int a(t)dt \quad \text{e} \quad x(t) = \int v(t)dt \quad (75)$$

Para computar essas grandezas, utilizamos integração numérica e optamos pela integração de Euler aqui. Um *loop* básico deste método, para nosso exemplo, é algo assim:

```
t = 0
while t < tmax do
    #determinamos força resultante do sistema
    #encontramos a aceleração com a segunda lei de Newton

    a = F/m
    novo estado: integramos a
    estado atual = novo estado
    t = t + h //h # o passo de tempo entre uma atualização
    #e outra, arbitrário
```

É importante dizer que, para utilizar Euler, precisamos assumir que as grandezas se mantêm constantes dentro de um passo de tempo e que a estrutura básica de atualização dos *loops* para a velocidade a posição são assim:

$$v^{[n+1]} = v^{[n]} + a^{[n]} * h \quad (76)$$

$$x^{[n+1]} = x^{[n]} + v^{[n]} * h \quad (77)$$

Note que não utilizamos exatamente a equação horária da posição do MUV (movimento uniformemente variado), como em (72), mas utilizamos a da velocidade (73).

Abaixo, segue um exemplo da simulação caso tivéssemos $x^{[0]} = 100$, $v^{[0]} = 0$ e $a(\text{constante, MUV}) = -10$:

```
v[0] = 0; x[0] = 100
t = 0; n = 0
while t < tmax do
    a[n] = -10
    v[n+1] = v[n] + a*h
    x[n+1] = x[n] + v*h
    n = n + 1; t = n*h
```

Adendos sobre a simulação

Deixando de focar um pouco nas grandezas envolvidas na colisão, vamos entender como funcionaria para simular uma animação mesmo.

Para isso, é necessária uma pequena adaptação no código:

```
v = []
v.append(0)
x = []
x.append(100)
a = []
```



```

t = 0
n = 0
tmax = 4
h = 1

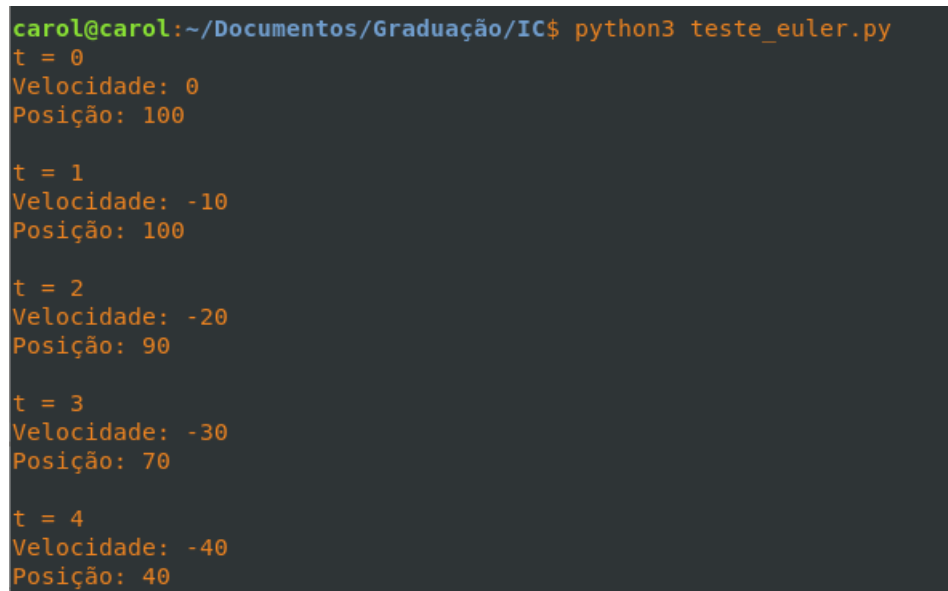
while(t <= tmax):
    print(v[n], x[n])
    a.append(-10)
    v.append(v[n] + a[n]*h)
    x.append(x[n] + v[n]*h)
    n += 1
    t = n*h

```

O `if` na linha 5 serve para atualizar a nossa animação na tela, antes da próxima iteração. Geralmente, vão ocorrer algumas iterações entre uma atualização do *frame* e outra.

Vamos implementar o código anterior, em `python`, e fazer a simulação com a integração de Euler para $t = (0, 1, 2, 3, 4)$ e as condições iniciais $x[0] = 100$, $v[0] = 0$ e $h = 1$ (lembrando que isso significa que estamos fazendo o *output frame* a cada iteração).

O resultado é esse:



```

carol@carol:~/Documentos/Graduação/IC$ python3 teste_euler.py
t = 0
Velocidade: 0
Posição: 100

t = 1
Velocidade: -10
Posição: 100

t = 2
Velocidade: -20
Posição: 90

t = 3
Velocidade: -30
Posição: 70

t = 4
Velocidade: -40
Posição: 40

```

Figura 10: Resultado da primeira simulação com a integração de Euler

Como exposto no livro, que fez o mesmo exemplo, existe uma diferença entre esses resultados obtidos com a simulação e aqueles que encontramos diretamente a partir do cálculo com as equações (72) e (73):

	t	0	1	2	3	4
Exact	v	0	-10	-20	-30	-40
	x	100	95	80	55	20
Euler $h = 1$	v	0	-10	-20	-30	-40
	x	100	100	90	70	40

Figura 11: Diferenças entre os resultados de uma simulação com Euler e o cálculo direto^[9]

Como podemos ver, a simulação da velocidade é uma aproximação perfeita, mas a posição oscila. Isso pode nos dar alguns problemas de precisão, então é necessário buscar formas de melhorar a simulação.

Uma das mais simples e a primeira a que recorreremos quando pensamos sobre integração e derivada é em aumentar o número de iterações entre um *frame* e outro. Relembrando a definição de derivada e a integração de Riemann, o raciocínio basicamente envolve analisar as operações fazendo limites de valores indo para zero, isto é, infinitesimalmente. No nosso caso, esse valor é representado pelo valor de h .

Vejamos o que ocorre com a simulação se a refizemos com $h = 0.5$:

Dessa forma, obtemos resultados um pouco mais próximos dos cálculos diretos do que com $h = 1$:

Então, se reduzirmos cada vez o passo entre uma interação e outra, chegaremos mais perto da resposta certa, mas ainda assim não será exatamente o resultado que queremos.

Resultado: O que a colisão causou no movimento, isto é, o que mudou no estado dos objetos depois dela?

Uma vez que sabemos que a colisão ocorreu e em que instante foi, podemos investigar como a velocidade da bola se comporta após a colisão, ou seja, o que a colisão causou no movimento da bola.

Lembrando que estamos lidando com uma situação muito simples, em que temos uma bola caindo em linha reta no chão (ou em alguma superfície plana).

Agora vamos analisar os efeitos de dois fatores na velocidade da bola após a colisão: elasticidade e atrito.

Elasticidade: a definimos como o "pulinho" que a bola dá quando colide com outra superfície, ou objeto e estudamos o efeito que esse fator tem na dissipação de energia do movimento (o quanto de energia é transferida e/ou conservada após a colisão).

Quando pensamos em uma bola caindo no chão, imaginamos o que vai acontecer com ela: vai "quicar", mais ou menos, dependendo do material de que ela é feita. Assim como quando brincamos com bolinhas de gude e sinuca, quando uma bola acerta outra, as duas têm seus movimentos alterados. A elasticidade tem a ver com essa análise, que envolve conceitos de momento linear e transferência de energia.

Existe uma propriedade chamada coeficiente de restituição c_r , que é o que usamos para medir, de certa forma, "a elasticidade" da colisão. Esse coeficiente nada mais é

```

carol@carol:~/Documentos/Graduação/IC$ python3 teste_euler.py
t = 0
Velocidade: 0.00
Posição: 100.00

t = 0
Velocidade: -5.00
Posição: 100.00

t = 1
Velocidade: -10.00
Posição: 97.50

t = 1
Velocidade: -15.00
Posição: 92.50

t = 2
Velocidade: -20.00
Posição: 85.00

t = 2
Velocidade: -25.00
Posição: 75.00

t = 3
Velocidade: -30.00
Posição: 62.50

t = 3
Velocidade: -35.00
Posição: 47.50

t = 4
Velocidade: -40.00
Posição: 30.00

```

Figura 12: Simulação de Euler com $h = 0.5$

	t	0	0.5	1	1.5	2	2.5	3	3.5	4
Exact	v	0		-10		-20		-30		-40
	x	100		95		80		55		20
Euler $h = 1$	v	0		-10		-20		-30		-40
	x	100		100		90		70		40
Euler $h = 0.5$	v	0	-5	-10	-15	-20	-25	-30	-35	-40
	x	100	100	97.5	92.5	85	75	62.5	47.5	30

Figura 13: Comparação do método com $h = 0.5$ e $h = 1$

que o fração da velocidade em que a bola estava logo antes da colisão que é preservada quando a bola retorna imediatamente após o movimento.

Quando c_r vale 1, significa que a bola voltou exatamente com a mesma velocidade em que colidiu. Nesse caso, temos conservação do momento linear e de energia. Isso acontece em casos onde consideramos que o movimento aconteceu em condições ideais, sem atrito, ou resistência do ar. Essas são as colisões elásticas.

Analogamente, temos as colisões inelásticas, nas quais c_r é próximo de 0. Nesse caso, nada do momento linear foi preservado e toda a energia foi dissipada.

Outra propriedade é o atrito, que tem um papel importante na dissipação de energia numa colisão. O atrito é uma força aplicada na direção tangencial à superfície de contato dos objetos e no sentido oposto ao movimento, ou seja, é uma força de resistência, que atrapalha o movimento.

Assim como na elasticidade, o coeficiente de atrito c_f vale 0 quando não há resistência alguma e 1 quando o atrito é bem forte.

6 Conclusões

Foi possível estudar os principais assuntos que introduzem a dinâmica do corpo rígido, bem como algumas simulações computacionais. Conseguimos explorar a física, a matemática e a programação em Python, o que possibilitou que a aluna estudasse tanto a teoria quanto que pudesse visualizar esses conceitos com implementações das simulações.

Estudamos os seguinte tópicos:

- Coordenadas globais e locais
- Matrizes de orientação
- Equações da posição e da velocidade
- Centro de massa
- Momento de inércia
- Atualização do estado de corpos rígidos
- Quaternions
- Introdução às colisões em duas dimensões
- Método de Euler

Além de algumas implementações destes assuntos.

Com isso, a aluna pôde enriquecer a experiência em programação em Python e com o mundo da pesquisa; apresentou seminários (presenciais e virtuais), teve contato com mestrandos e doutorandos da área de Computação Gráfica e aprimorou suas habilidades de escrita e de pesquisa científicas.

7 Referências

[1] HOUSE, D.H. ; KEYSER, J.C. **Foundations of physically based modeling and animation**. 20017 – CRC Press Taylor & Francis Group, Boca Raton, 2017.

[2] GIESSEN, Fabian. **Quaternion differentiation** . Wordpress, 2014. Disponível em:

<https://fgiesen.wordpress.com/2012/08/24/quaternion-differentiation/>

[3] WIKIPEDIA. **Second Moment of Area**. Disponível em:

https://en.wikipedia.org/wiki/Second_moment_of_area

[4] BULVIDOVICH, P.V. **The Inertia Tensor**. Disponível em:

http://www.lattice.itep.ru/~pbaivid/lecture_notes/Inertia_Tensor_lecture_notes.pdf

[5] SCHULZ, Adriana **Rotations and Interpolations**. 2010. Disponível em:

<http://w3.impa.br/~aschulz/anim/rotations.pdf>

[6] SARABANDI, Soheil; THOMAS, Federico. **Accurate Computation of Quaternions from Rotation Matrices**. Disponível em:

<http://www.iri.upc.edu/files/scidoc/2068-Accurate-Computation-of-Quaternions-from-Rotation-Matrices.pdf>

[7] JIA, Yan-Bin. **Quaternions and Rotations**. 2013. Disponível em:

<http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf>