Física - Movimento angular

Introdução

Nosso mecanismo de física agora pode mover objetos de maneira fisicamente precisa - pelo menos em linhas retas. Além do movimento linear, os corpos rígidos também podem girar, mudando sua orientação ao longo do tempo. Neste tutorial, veremos como aplicar as forças corretas para permitir que esse movimento rotacional aconteça e ajustar o código de empurrar objetos que introduzimos no tutorial anterior para nos permitir girar e girar os objetos conforme eles são clicados com o mouse .

Novos Termos

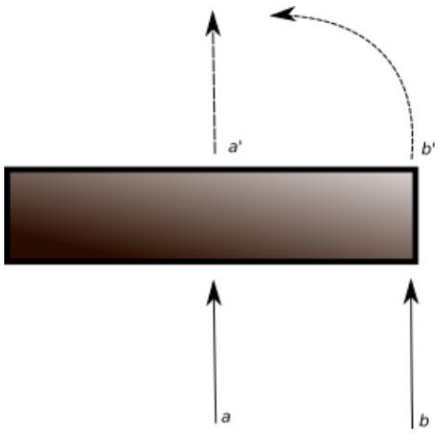
Torque

Vimos no tutorial anterior que as forças transmitidas a um objeto resultam em uma mudança de velocidade, com a magnitude da mudança (a aceleração) sendo relacionada à (inversa) do objeto.
massa:

uma = Fmÿ1

A partir desta equação, podemos ver como alterar a velocidade linear de um objeto; como ele se move em linha reta. À medida que as simulações físicas que desejamos criar se tornam mais complexas, provavelmente também queremos que os objetos girem e girem ao colidir com outros objetos. Para fazer isso, precisamos determinar quanta velocidade angular o objeto físico possui e integrá-la de acordo com a aceleração angular.
Assim como a aceleração linear é aplicada por meio de força, a aceleração angular é aplicada por meio de torque.

Podemos, se quisermos, adicionar torque diretamente aos objetos, adicionando uma força angular a cada quadro - as rodas de um carro podem ser modeladas desta forma. As coisas ficam mais interessantes se considerarmos os outros casos em que o movimento de torção poderia ser aplicado a um objeto. Considere os dois casos a seguir, de um objeto cubóide sendo 'empurrado' por uma força aplicada nos pontos a ou b:



Como o objeto se moveria sob cada uma das forças mostradas? Supondo que estamos lidando com um corpo rígido, com uma distribuição uniforme de massa ao longo de seu volume, é bastante intuitivo que se o objeto tivesse uma força aplicada no ponto a, seu movimento resultante seria semelhante a a' - ele se moveria uma linha reta. Porém, no ponto b, o objeto deve girar, com o canto do cubóide seguindo um caminho semelhante a b'. Você pode testar isso movimentando uma caneta em sua mesa. Você verá que quanto mais longe do meio da caneta você movimentar, mais a caneta deseja girar à medida que gira.

se move, então nosso cuboide de exemplo deveria realmente começar a girar - a força aplicada adicionou torque ao objeto, resultando em velocidade angular, bem como em velocidade linear.

Determinar a quantidade de torque que uma força aplica a um objeto é bastante simples. Se estivermos aplicando uma força F em uma posição relativa de d a partir do centro de massa do objeto, a quantidade de torque ÿ é definida simplesmente como:

$$ÿ = d \times F$$

Onde × é o produto vetorial entre os dois vetores. Como você viu no módulo gráfico anterior, o produto vetorial entre dois vetores produz um vetor que é ortogonal a ambos - neste exemplo, como d é direcionado ao longo do eixo x e F ao longo do eixo z, ele produzirá um vetor direcionado ao longo o eixo y. Isto resulta numa quantidade de rotação em torno deste eixo, fazendo com que o nosso objeto de exemplo gire, alterando a sua guinada, numa quantidade proporcional à magnitude do vetor de torque e à massa do objeto.

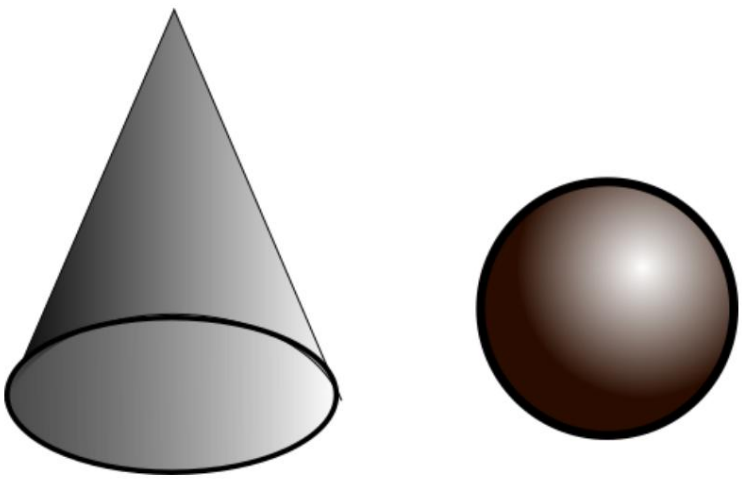
Assim como a unidade de força é o Newton, a unidade de torque é o Newton Metro – o resultado da aplicação de uma força a 1 metro do centro de massa do objeto.

Inércia

A aceleração de um objeto está relacionada à força que atua sobre ele e à massa do objeto. Dado isso, o torque de um objeto não deveria ser influenciado também por sua massa? Na verdade deveria! No entanto, não é tão fácil como com o movimento linear. Quando empurramos um objeto em seu centro de massa (ou seja, aplicamos uma força puramente linear), estamos tentando mover todo o objeto de uma só vez e na mesma velocidade. Quando empurramos um objeto com alguma força rotacional (como no exemplo cubóide anterior), a quantidade de força necessária para girar o objeto em um eixo específico depende da distribuição da massa em seu volume - estamos tentando fazer com que a massa em torno do as partes externas desse cubóide se movem mais do que a massa em seu meio. Portanto, para determinar a quantidade de movimento angular que uma força deve transmitir a um objeto, precisamos de mais do que apenas um valor de massa escalar, mas de algo que possa descrever a distribuição dessa massa em torno do volume do objeto em cada eixo - o momento de inércia. Esta quantidade descreve o quão resistente um objeto é às mudanças em sua velocidade angular, assim como a massa limita as mudanças na velocidade linear.

Tensor de Inércia

Podemos representar o momento de inércia de um objeto usando um tipo especial de matriz, conhecido como tensor. Para entender por que não podemos ter apenas um único valor escalar para o nosso momento de inércia, considere as seguintes formas:



Se assumirmos que ambas são formas sólidas, feitas de um material único e uniforme, então deve ficar claro que o cone tem uma distribuição de massa diferente (principalmente em torno da parte inferior da forma) em comparação com a esfera (distribuída uniformemente).

Momentos comuns de inércia

Para objetos simétricos simples, normalmente não precisamos fazer nada muito trabalhoso para determinar o tensor de inércia; Há uma série de cálculos padrão para objetos com formatos semelhantes aos volumes de colisão padrão que usaríamos em uma simulação de física de jogo.

Esfera Sólida

$$I = \frac{2}{5}mr^2 \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix}$$

Cubóide Sólido

O tensor de inércia para um cubóide é um pouco mais complexo, pois a distribuição da massa depende da largura, altura e comprimento do cubóide. Podemos representar isso usando o seguinte tensor de inércia:

$$\begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

Onde I_x, I_x , e I_x são os seguintes:

$$I_x = \frac{1}{12}m(z^2 + y^2)$$
$$I_y = \frac{1}{12}m(x^2 + z^2)$$
$$I_z = \frac{1}{12}m(x^2 + y^2)$$

Como antes, m é a massa do objeto e x, y e z são a largura, altura e comprimento do cubóide.

Cone Sólido

Embora seja uma forma bastante incomum, pode ser útil dar uma olhada mais de perto no exemplo do cone anterior - exatamente como a força angular muda dependendo da forma do objeto? Primeiramente, vamos dar uma olhada no momento de inércia de um cone, que assume o ponto central do objeto como o ponto do cone:

$$\begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

Onde I_x, I_x , e I_x são os seguintes:

$$I_x = I_z = \frac{3}{20}mr^2 + \frac{1}{10}mh^2$$

$$I_y = \frac{3}{10}Mr^2$$

Então, se a massa do nosso cone for 1, sua base tiver um raio de 1 e seu pico tiver 1 unidade de altura, obteremos o seguinte tensor:

$$\begin{bmatrix} 0{,}25 & 0 & 0 \\ 0 & 0{,}3 & 0 \\ 0 & 0 & 0{,}25 \end{bmatrix}$$

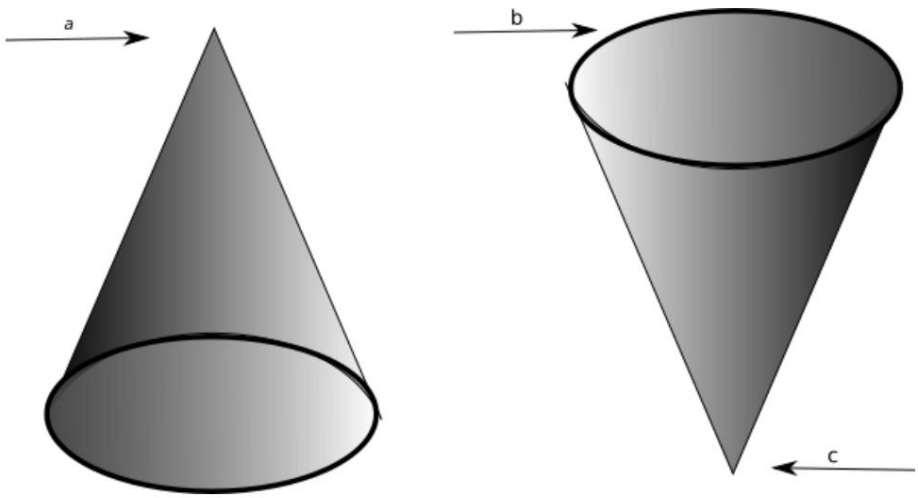
Embora possamos ver visualmente que um cone tem mais massa distribuída ao longo de x e z, podemos agora ver como isto resulta numa matriz de escala com diferentes quantidades de escala nos eixos onde a maior parte da massa está situada na forma.

Tensor de inércia inversa

No tutorial anterior, vimos que, em vez de apenas massa, poderíamos economizar computação (e obter a capacidade de representar naturalmente objetos de 'massa infinita' que não queremos mover) usando a massa inversa de um objeto em nossos cálculos . Pelas mesmas razões, podemos fazer o mesmo com o momento de inércia e, em vez disso, criar um tensor de inércia inverso I acima, isso é ÿ1 . Para momentos de inércia como o tão simples quanto usar o inverso de cada um dos valores abaixo da diagonal do tensor.

Girando o Tensor de Inércia

Você pode estar olhando para os momentos de inércia codificados como tensores acima e ainda estar se perguntando exatamente por que estamos usando uma matriz para representá-los, em vez de um vetor, se todos os valores terminam na diagonal da matriz. Há duas razões: por um lado, nosso objeto pode não ser simétrico e, portanto, requer um cálculo muito mais complexo usando integrais (imagine fatiar o objeto assimétrico em pequenos cubos e ver qual proporção desses cubos está ao longo de cada eixo 3D, e depois, em cada eixo, como essas proporções mudam quando se movem entre si), isso resulta em uma matriz 3x3 que descreve a distribuição entre cada eixo. Em segundo lugar - aplicamos forças no espaço mundial (ou seja, uma força de (10,0,0) deveria empurrar um objeto ao longo do eixo x global), mas o momento de inércia descreve a distribuição em um espaço local ao do objeto . Para objetos sem uma distribuição uniforme de massa (como nosso exemplo de cone anterior), isso se torna importante, pois sua resposta rotacional às forças deve ser consistente - a mesma força relativa aplicada a esse cubóide deve ter o mesmo efeito, independentemente da orientação do cubóide. é em:



As forças a e c deveriam produzir o mesmo efeito no cone (assumindo que tenham a mesma magnitude), enquanto a força b, mesmo que tivesse a mesma magnitude que as outras, deveria ter menos efeito sobre o cone, pois a distribuição de massa em todo o seu volume é diferente.

Uma maneira de resolver isso seria, para cada força aplicada a um objeto em um referencial, transformar essa força pelo inverso da orientação do objeto (para trazê-lo para o espaço local) e depois multiplicá-la pelo tensor para determinar o efeito de inércia, antes de multiplicar o resultado de volta ao espaço mundial, multiplicando novamente pela orientação. Isso funciona... mas e se houver muitas forças adicionadas ao objeto em um quadro? São muitas transformações de espaço para aplicar! Em vez disso, podemos girar o tensor de inércia do objeto pela orientação do objeto uma vez por quadro, trazendo-o do espaço local do objeto para o espaço mundial de nossa simulação e, portanto, adequado para quantas forças serão aplicadas ao objeto nesse quadro.

Velocidade angular

Agora que entendemos por que temos uma matriz para o tensor de inércia, podemos ver como usá-la para realmente determinar quanta velocidade angular ganhamos com o torque. É exatamente o mesmo processo que com a velocidade linear - exceto que desta vez multiplicamos o torque pelo tensor interina inverso (em vez de

massa inversa por aceleração). Da nossa quantidade de força de torque ÿ , podemos determinar a quantidade de aceleração angular ÿ, e então integre isso na velocidade angular ÿ:

$$\ddot{y} = eu\ \ddot{y}^{\ddot{y}1}$$
$$\ddot{y} = \ddot{y}dt$$

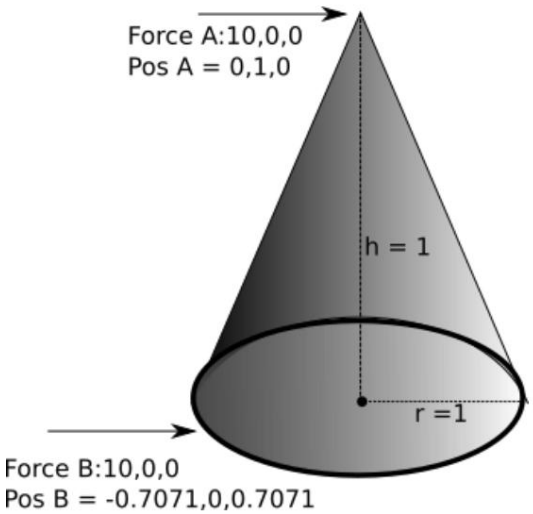
O número de símbolos está aumentando! No código, porém, tudo isso é bastante fácil de implementar, e no código do tutorial podemos continuar usando nomes mais descritivos para cada um dos conceitos à medida que eles são expandidos em nosso mecanismo de física completo.

Exemplo de cone

Para obter uma imagem mais clara do uso do tensor de inércia e como ele altera a aceleração angular aplicada devido ao torque, vamos revisitar o tensor de inércia cônico descrito anteriormente e seu inverso:

$$eu = \begin{matrix} \ddot{y} \\ \ddot{y} \end{matrix} \begin{matrix} 0{,}25\ 0\ 0 \\ 0\ 0\ 0{,}25 \end{matrix} \begin{matrix} 0 \\ 0 \end{matrix} \quad \begin{matrix} 0{,}3\ 0 \\ \ddot{y} \end{matrix} \qquad \ddot{y}_{EU}{}^{\ddot{y}1} = \begin{matrix} \ddot{y} \\ \ddot{y} \end{matrix} \begin{matrix} 4\ 0 \\ 0\ 3{,}333\ 0 \\ 0\ 0 \end{matrix} \begin{matrix} 0 \\ \ddot{y} \\ 4 \end{matrix} \begin{matrix} \\ \\ \ddot{y} \end{matrix}$$

Agora vamos ver como esse tensor mudaria a quantidade de torque aplicado em dois pontos diferentes, mostrado neste diagrama:



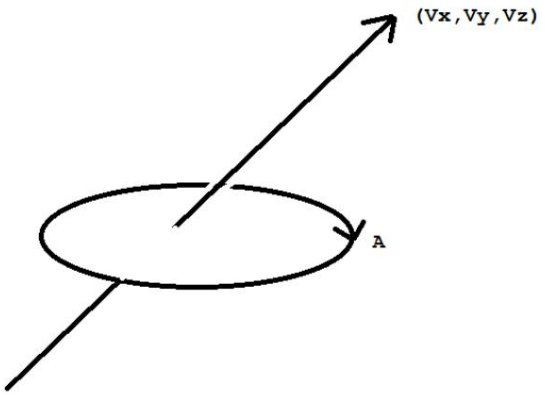
Cada força está sendo aplicada em um ponto a 1 unidade de distância da posição do objeto, apenas ao longo de um eixo diferente. Usando esses vetores, podemos ver que se aplicássemos uma força no ponto A, obteríamos um torque resultante de (0,0,10) (o resultado do produto vetorial da força a pela posição a). No ponto B obteríamos um torque de (0,0,-7,071), dando-nos dois eixos de rotação muito diferentes e diferentes quantidades de rotação em torno desses eixos. Até agora, esses cálculos não usaram nosso tensor de inércia e, portanto, não podem descrever quanta rotação esses torques realmente aplicarão - a massa resistirá às tentativas de alterar o momento do objeto, portanto a distribuição da massa é importante. Depois de transformar cada um desses vetores pelo tensor de inércia acima, obtemos a aceleração angular resultante ÿ:

$$\text{Resultado da força A} = (0, 0, 40) = I^{\ddot{y}1}\ (0, 0, 10)$$
$$\text{Resultado da força B} = (0, \ddot{y}23{,}6, 0) = I^{\ddot{y}1}\ (0, \ddot{y}7{,}071, 0)$$

Embora ambas as forças aplicadas tenham a mesma magnitude, elas foram aplicadas em posições diferentes no volume e, portanto, resultaram em diferentes quantidades de torque, que quando dimensionadas pelo tensor de inércia inversa anterior, nos permitem ver que a força aplicada no ponto A causa muito mais rotação do que a força aplicada no ponto B - há menos massa nesse ponto para resistir à mudança.

Quatérnios

No módulo gráfico, se você examinou o código da animação esquelética, você encontrará quatérnios. Um quaternion é uma forma eficiente de armazenar uma orientação, que também possui a propriedade útil de um método bem formado para girar uma orientação. Ao contrário de uma matriz de rotação, que teria 9 elementos (ou 16 se fosse homogênea), um quaternion pode armazenar uma orientação usando apenas 4 valores. Ele faz isso codificando dentro dele um eixo e a rotação em torno dele:



Para integrar a velocidade angular na orientação de um objeto, precisamos formar um quatérnio que represente a quantidade pela qual desejamos girar e, em seguida, usar a multiplicação do quatérnio para formar um novo quatérnio que é a combinação da orientação antiga e da quantidade a ser movida. por. Você pode, portanto, pensar que os 4 valores seriam um vetor de direção normalizado para representar o eixo e um escalar para representar a rotação em torno dele, mas é um pouco mais complicado do que isso. Queremos uma forma de representar uma orientação e também uma mudança nessa orientação, por isso precisamos de uma forma de combinar e alterar o eixo e o ângulo de uma forma consistente. Em vez do eixo e do ângulo diretamente, os 4 valores de um quatérnio são formados a partir de um vetor V e do ângulo A da seguinte forma:

$$Qx = \text{sen}(A/2) \ddot{y} V x$$
$$Qy = \text{sin}(A/2) \ddot{y} V y$$
$$Qz = \text{sen}(A/2) \ddot{y} V z$$
$$Qw = \cos(A/2)$$

Este formulário possui algumas propriedades úteis. Pois, podemos inverter a orientação do quatérnio Q tomando seu conjugado:

$$Q\ddot{y}1 = [\ddot{y}Qx , \ddot{y}Qy , \ddot{y}Qz , Qw]$$

Assim como uma matriz de transformação, o inverso de um quatérnio nos move de volta para o outro lado - se girando um vetor pelo quatérnio Q e depois pelo quatérnio Qÿ1 , voltaríamos ao ponto de partida.

Para construir um quatérnio Q3 que codifique uma transformação de Q1 e Q2, devemos multiplicar esses quatérnios, da mesma forma que acontece com as matrizes. A multiplicação de quatérnios fica assim:

$$P^{x}_{3} = (Q\,x_1 \cdot Q^{c}_{2}) + (Q\,w_1 \cdot Q^{x}_{2}) + (Q^{e}_{1} \cdot Q^{z}_{2}) \ddot{y} (Q\,z_1 \cdot Q^{y}_{2})$$
$$P^{anoo}_{3} = (Q^{sin}_{1} \cdot Q^{c}_{2}) + (Q\,w_1 \cdot Q^{sin}_{2}) + (Q\,z_1 \cdot Q^{x}_{2}) \ddot{y} (Q\,x_1 \cdot Q^{z}_{2})$$
$$P^{z}_{3} = (Q\,z_1 \cdot Q^{c}_{2}) + (Q\,w_1 \cdot Q^{z}_{2}) + (Q\,x_1 \cdot Q^{sin}_{2}) \ddot{y} (Q^{e}_{1} \cdot Q^{y}_{2})$$
$$P^{c}_{3} = (Q\,w_1 \cdot Q^{c}_{2}) \ddot{y} (Q\,x_1 \cdot Q^{y}_{2}) \ddot{y} (Q^{e}_{1} \cdot Q^{anoo}_{2}) \ddot{y} (Qz_1 \cdot Q^{z}_{2})$$

Como você pode perceber pela fórmula acima, a multiplicação de quatérnios não é comutativa, ou seja, Q1 x Q2 não nos dá a mesma resposta que Q2 x Q1 - isso deve ser familiar para você, pois as matrizes de transformação funcionam da mesma maneira.

No tópico de matrizes de transformação, é possível expandir um quatérnio em uma matriz de rotação 3x3 e, portanto, também em uma matriz homogênea 4x4 completa, de modo que não 'perdemos' nada armazenando orientações como um quatérnio, mas fazemos obtenha uma maneira fácil de mantê-lo separado da escala de um objeto.

Integrando velocidade angular

A última coisa que precisamos fazer com um quatérnio é ver como transformar uma representação do ângulo do eixo em um quatérnio. Por que? Pense na nossa velocidade angular anterior - nós a armazenamos como um vetor, que representa o quanto o objeto está girando em torno desse eixo. Precisamos fazer duas coisas: primeiro, precisamos descobrir quanto essa velocidade angular acrescenta por intervalo de tempo e, em segundo lugar, precisamos transformar o vetor do eixo angular em um quatérnio. Podemos fazer as duas coisas ao mesmo tempo, com as seguintes operações:

$$Vtemp = 2^{\frac{dt(velocidadeang)}{}}$$
$$Qtemp = [Vtempx, Vtempy, Vtempz, 0] \cdot orientação$$
$$orientação = orientação + Qtemp$$

Vtemp mantém nossa mudança relativa na orientação para o intervalo de tempo atual, e Qtemp transforma isso no que é conhecido como quaternion puro - um sem quarto elemento, em relação à nossa orientação original. Parece muito confuso e depende da propriedade multiplicativa dos quatérnios, além de uma normalização, para funcionar. Quaternions são uma daquelas coisas que, como engenheiros de jogos, gostamos devido à sua eficiência, mas também não gostamos devido à natureza obtusa de sua operação - não precisaremos fazer mais nada com eles neste módulo, então tudo o que precisamos lembrar é que eles são representações eficientes de orientação.

Código do Tutorial

Para demonstrar o movimento angular em nossos corpos rígidos, vamos modificar o tutorial anterior, para que as forças aplicadas no clique do mouse transmitam a quantidade correta de torque para a posição em que clicamos; isso deve permitir que as formas girem na tela.

Torque de integração

Para ajustar a orientação do nosso objeto ao longo do tempo, precisamos integrar o torque aplicado na velocidade angular do objeto. Para fazer isso, vamos modificar a função IntegrateAccel, adicionando o seguinte código após a chamada SetLinearVelocity que adicionamos no tutorial anterior:

```
1 // Começamos a adicionar novo código após esta linha existente:
2               objeto -> SetLinearVelocity (linearVel); //código anterior
3
4               // Coisas angulares
5               Torque Vector3 = objeto -> GetTorque();
6               Vector3 angVel=objeto->GetAngularVelocity();
7
8               objeto -> UpdateInertiaTensor(); //atualiza tensor vs orientação
9
10              Vector3 angAccel=objeto->GetInertiaTensor()*torque;
11
12              angVel += angAccel * dt ; //integra aceleração angular ! objeto -> SetAngularVelocity
13              (angVel);
14      }
15}
```

Método PhysicsSystem::IntegrateAccel

Assim como acontece com a velocidade linear, calculamos um valor de aceleração a partir da massa inversa do objeto - exceto que desta vez estamos usando o tensor de inércia do objeto para transformar a aceleração (linha 10). Como parte deste processo, também atualizaremos o tensor de inércia do objeto, com base na sua orientação atual - lembre-se, precisamos girar o tensor de inércia para que seus valores reflitam com precisão a orientação atual do objeto. Feito isso, podemos apenas atualizar a velocidade angular com a aceleração angular atual, escalonada pelo timestep dt (linha 12).

Integrando velocidade angular

Com a aceleração angular integrada, podemos mudar o método IntegrateVelocity para atualizar corretamente a orientação do objeto de acordo com sua velocidade angular. No IntegrateVelocity método, adicione o seguinte código após a chamada para SetLinearVelocity:

```
1 // Começamos a adicionar novo código após esta linha existente:
2                 objeto -> SetLinearVelocity (linearVel);
3
4                 // Coisas de orientação
5                 Orientação quaternion = transformação. GetLocalOrientation();
6                 Vector3 angVel=objeto->GetAngularVelocity();
7
8                 orientação = orientação +
9                         (Quaternion(angVel*dt*0,5f orientação. Normalizar()); ,     0,0 f ) * orientação );
10
11
12                transformar. SetLocalOrientation(orientação);
13
14                //Atenua também a velocidade angular
15                angVel = angVel * frameDamping;
16                objeto -> SetAngularVelocity (angVel);
17        }
18}
```

Método PhysicsSystem::IntegrateVelocity

A integração real ocorre na linha 9. O 0,5 é devido à forma como os quatérnios funcionam - confie que não estamos 'perdendo' qualquer velocidade angular. A maior parte desse método está centrada na entrada e saída de dados orientação quaternion Assim como acontece com a velocidade linear, também aplicaremos algum amortecimento (linha 15), então que os objetos não giram para sempre.

Aplicando forças

Para adicionar torque a um objeto de física, precisaremos de um novo método na classe PhysicsObject, AddForceAtPosition:

```
1 void PhysicsObject :: AddForceAtPosition (
2                         const Vector3 e força adicionada                    ,    const Vetor3 & posição) {
3         Vector3 localPos=posição -transformação->GetWorldPosition();
4
5         força += força adicionada;
          torque += Vetor3 :: Cruzado (localPos                          ,    adicionadaForça);
6 7}
```

Método PhysicsObject::AddForceAtPosition

Isso demonstra em código o cálculo do torque - a partir da variável de posição mundial passada, precisamos calcular a posição relativa ao centro de massa do objeto (linha 3) e, em seguida, usar a cruz produto para determinar o eixo em torno do qual essa força fará o objeto girar. Lembre-se disso o produto vetorial produz valores invertidos dependendo da ordem dos parâmetros, portanto, tome cuidado ao usando isso.

Método MoveSelectedObject

Para usar o novo método AddForceAtPosition, precisamos modificar o MoveSelectedObject da classe Tu-torialGame. Em vez de chamar AddForce, vamos chamar AddForceAtPosition e passe o ponto de colisão no mundo como o segundo parâmetro:

```
if (mais próximoCollision. node == objeto de seleção) {
        selectionObject->GetPhysicsObject()-
                >AddForceAtPosition(ray.GetDirection()*forceMagnitude,closeCollision.collidedAt);
}
```

Método TutorialGame::MoveSelectedObject

Conclusão

Se clicarmos nos objetos agora, eles deverão ser empurrados de modo que girem levemente sob a aplicação de torque. Isto depende de onde os objetos são clicados - lembre-se, quanto mais longe do centro de massa clicamos, mais a força que transmitimos ao objeto é aplicada como um movimento de torção.

Nossos objetos estão começando a agir um pouco mais como objetos “reais” agiriam, mas eles não são perfeitos! Atualmente podemos empurrar objetos para o chão e uns para os outros, sem consequências. Na próxima parte da série de tutoriais, veremos como remediar isso usando métodos de detecção e resolução de colisões, que juntos nos permitem determinar onde um objeto tocou outro e o que isso fará com o movimento linear e angular dos objetos em colisão.

Trabalho adicional

1) Para acentuar o efeito do torque em nossas formas e demonstrar como o tensor de inércia afeta nossos objetos, tente criar formas cúbicas que não sejam uniformes em cada eixo e tente aplicar forças a elas em pontos variados com cliques do mouse - você deve ser capaz de fazer os objetos parecerem 'mais pesados' com uma mudança na massa, e uma força aplicada em torno de alguns eixos deve ter um resultado visivelmente diferente de outros.

2) As esferas ocas têm um tensor de inércia diferente das esferas sólidas (sua massa é distribuída longe da origem). Investigue o que pode ser esse tensor de inércia e tente modificar o método Tutorial-Game::AddSphereToWorld para permitir escolher se a esfera é oca ou sólida.