Simulação da Animação de Corda baseada em Física

Daniel Guimarães – 1910462
Mariana Barreto – 1820673
Mateus Levi – 1710954

Introdução

O projeto busca realizar uma simulação de um pedaço de corda em 2D considerando os efeitos gravitacionais. A corda será representada por uma grade de partículas com barras de restrição utilizando o método numérico conhecido como Integração de Verlet. Para tal método, foi utilizada a relaxação para tratar as restrições.

Desenvolvimento

Para realizar a simulação, foi escolhida a linguagem de programação *Python*. Além disso, o pacote *pygame* foi utilizado para a visualização da corda e a biblioteca *numpy* foi utilizada para criar vetores necessários para implementação do método. Os vetores foram criados como *numpy.array* ao invés de listas para facilitar as operações vetoriais.

Em primeiro lugar, todas as partículas são inicializadas. Cada partícula contém as informações da posição anterior da partícula, posição atual da partícula e um *status* que indica se a partícula está fixa ou não. Quando as partículas são inicializadas, todas elas não são fixas com exceção da primeira. A posição anterior de cada partícula é calculada a partir de:

```
previous_pos = np.array([0.1 * i, tam_corda - dist_minima * i + 10])
```

Ou seja, a coordenada x será equivalente a 10% da posição da partícula no vetor, enquanto a coordenada y será equivalente à diferença entre o tamanho da corda e a distância mínima estabelecida entre as partículas. Isso foi feito com o intuito de gerar partículas que formassem a corda "sequencialmente" e de forma que a corda, em sua posição inicial, estivesse prestes a iniciar movimento pendular (estando elevada e ligeiramente à direita do eixo central). Tanto o tamanho da corda quanto a distância mínima são parâmetros que podem ser alterados.

Depois de criar as partículas, as distâncias iniciais entre essas partículas são calculadas e guardadas em um vetor e, a partir da posição anterior obtida anteriormente, a posição atual é calculada. O passo de Euler é tomado para obter essa primeira posição, isto é:

```
current_pos = particles[i].previous_pos + h * v0
```

A posição atual inicial de cada partícula corresponderá à posição anterior dessa partícula acrescida do passo de integração multiplicado pela velocidade inicial. A velocidade inicial $\mathbf{v_0}$ utilizada foi de (0, 0), efetivamente mantendo as posições iniciais.

Por fim, o relaxamento inicial é feito. Um relaxamento inicial é feito entre cada partícula e sua partícula com posição imediatamente superior. Por exemplo, se temos apenas duas partículas p_A e p_B com uma distância d_A, o relaxamento é feito em cima de p_A e p_B considerando d_A. Já o segundo relaxamento feito considera uma partícula de posição posterior a p_B. Ou seja, se fosse acrescida uma partícula p_C em uma posição imediatamente posterior a p_B com uma distância d_B, o segundo relaxamento seria feito em cima de p_A, p_C considerando a distância entre essas duas partículas ($d_A + d_B$).

Para cada relaxação, é calculada distância entre as posições atuais de cada partícula. Um ajuste é feito em cima da diferença dessa distância e da distância considerada para a relaxação. A direção de cada partícula é calculada dividindo as duas distâncias. Dessa forma, a nova posição das partículas será incrementada de acordo com:

```
point1.current_pos = (adjust / 2) * direction
point2.current_pos = (adjust / 2) * (-direction)
```

É feita uma exceção para o caso das partículas que são fixas, pois apenas as partículas móveis são ajustadas. A diferença do ajuste nesse caso é que como a partícula fixa não move, a partícula móvel tem que se ajustar por ela. Por isso, a partícula móvel tem sua posição incrementada pelo produto do ajuste com a direção, e não pela metade como nos demais casos.

Após essa inicialização, todos pontos móveis têm sua posição atualizada considerando o seguinte cálculo:

```
next_pos = current_pos + (1 – delta) * (current_pos – previous_pos) * ((h * h)/m)
* fg
```

Esse cálculo é a aplicação do Método de Integração de Verlet para a simulação física de partículas no espaço 2D ou 3D. O *delta* representa o coeficiente de amortecimento δ, o *h* equivale ao passo de integração, *m* diz respeito à massa e *fg* à força da gravidade. Nesse caso, o trabalho não leva em consideração a força do vento. A força da gravidade considerada para as simulações é de 9.8 m/s^2.

Após obter as próximas posições que correspondem às novas posições atuais das partículas, é feito a relaxação em cima dessas novas posições. O procedimento de relaxação é o mesmo que o indicado acima e ele é feito para cada barra.

As barras do sistema incluem tantas barras “reais” que compõem a corda de fato quanto barras “imaginárias” entre partículas não-adjacentes que formam a corda, auxiliando na manutenção das distâncias entre elas e, consequentemente, na não-deformação do sistema com o tempo.

Essa seção que diz respeito à atualização da posição das partículas utilizando o método de Verlet é feito indefinidamente. O objetivo é verificar o movimento pendular da corda, de forma que eventualmente a corda se encontrará no centro e não terá mais sua posição atualizada.

Para fazer a parte gráfica, foi necessário escolher uma taxa de atualização dos pontos, por exemplo, 60 atualizações por segundo, o que significa que todos os cálculos e renderizações precisam ser feitos em menos de 1/60 segundos. No caso em que os cálculos e renderização sejam feitos em tempo suficiente, o programa espera até completar esse intervalor de tempo, para fazer a próxima iteração. Isso foi feito para que a simulação possa fluir na taxa de atualização correta, mesmo que fosse possível executar mais rápido

Resultados e Análise

Abaixo está um exemplo de execução do programa:

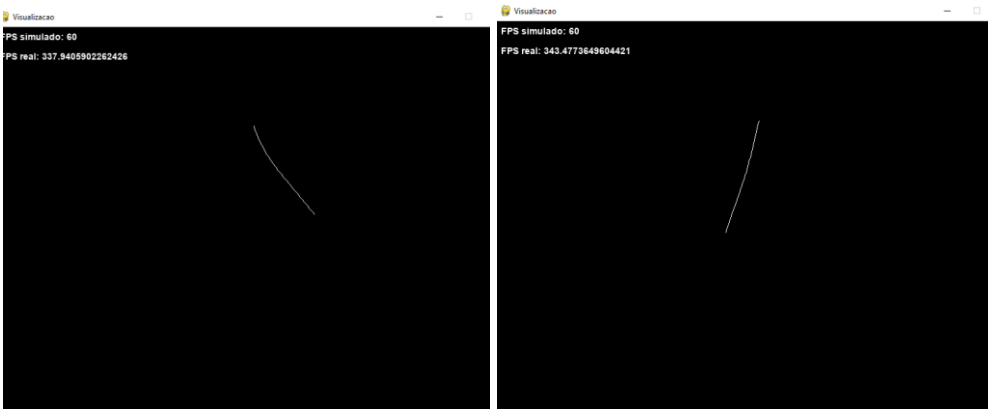

Figura 2 e 3: Capturas sequenciais da simulação feita pelo código

Assim que inicia o programa, a corda, que iniciou inclinada para direita, começa a realizar seu movimento pendular de maneira realista, conforme o esperado. Após um determinado tempo, a corda começa a se estabilizar, perdendo seu movimento, até parar na vertical.

Também é possível observar os efeitos da massa da corda na simulação. Ao aumentar o peso da corda, ela começa a perder sua flexibilidade, e ao diminuir o peso, ela de fato aparenta se movimentar mais levemente, com uma alta flexibilidade.

Número de Partículas	Número de Relaxações	FPS máximo
20	112	610
40	252	270
80	532	140
100	672	115

Figura 4: Tabela com informações relativas ao número de relaxações

Para várias simulações diferentes, como pode ser visto na figura 4, o tempo de processamento do movimento da corda sempre foi bem pequeno. Na figura 1 por exemplo, a simulação estava fazendo 10 atualizações por segundo, mas estava fazendo um processamento que permitia executar até 200 atualizações por segundo.

O número de barras para o relaxamento varia, então ao executar a simulação, o programa avisa quantas relaxações serão executadas em cada iteração, como pode ser visto na figura abaixo:

```
pygame 2.1.0 (SDL 2.0.16, Python 3.9.8)
Hello from the pygame community. https://www.pygame.org/contribute.html
NUMERO DE PARTICULAS: 40
NUMERO DE BARRAS (RELAXACOES POR ITERAÇÃO): 252
```

Figura 5: Informações exibidas

Para a simulação do movimento pendular da corda ficar convincente independentemente dos parâmetros usados (como massa, tempo do passo, entre outros), foram testadas diversas quantidades de barras (e, consequentemente, relaxações).

O que aconteceu foi que testes com quantidades maiores de partículas deixaram claro que a qualidade da simulação deteriorava com um número baixo de relaxações, principalmente quando a massa das partículas era um valor baixo (por exemplo, menor que 1).

Para solucionar esse problema, o número de relaxações (mais especificamente, de barras criadas) foi aumentado progressivamente. Isso significou criar mais "adjacências" entre as partículas da corda, visando evitar deformação da mesma por causa do próprio relaxamento sob poucas restrições e pelo movimento mais intenso de partículas com poucas massas.

Finalmente, o valor ótimo encontrado para o número de barras foi da ordem de $7n$, onde n representa a quantidade de partículas no sistema. Mais especificamente, a simulação mostrou resultados convincentes com um valor de $7n - \sum_{i=1}^{7} i$ relaxamentos por iteração.

Na prática, isso significa que nossa implementação apresenta os melhores resultados quando cada partícula possui barras adjacentes a até 7 outras partículas – restringindo então sua posição em função dos relaxamentos, que manterão uma distância prévia da partícula em relação as outras.

O sistema implementado em seu estado final está genérico o bastante para representar qualquer sistema de partículas "lineares" – isso é, que possam ser consideradas como formando uma corda.

Abordando a implementação de maneira mais específica: a classe `CordaSimul`, a partir de um conjunto de coordenadas (x,y) e de uma lista de valores de massa, forma a corda criando objetos da classe `Particle` (que são definidos por uma posição e um valor de massa) e gerando barras de adjacência a partir de instâncias da classe `Bar`. A partir da função `proxima_avaliacao`, a simulação física é implementada pela Integração de Verlet e pelo uso da técnica de relaxamento.

Pelo uso das classes `Bar` e `Particle` mencionadas, a implementação poderia ser estendida para simular também outras configurações de sistemas massa-barra de partículas; por exemplo, um tecido na forma de uma malha de partículas massivas.

Para isso, seria necessário fazer algumas adaptações no código: por exemplo, o código de criação de barras deveria ser alterado para criar barras em todos os sentidos de adjacência e não apenas de forma linear como na implementação atual. De qualquer forma, os fundamentos para essa extensão de funcionalidades do sistema já estão disponíveis na implementação atual.