UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

RIO DE JANEIRO
2025

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Cláudio Esperança

RIO DE JANEIRO

2025

CIP - Catalogação na Publicação

R484t

Ribeiro, Tatiana de Sousa
Titulo / Tatiana de Sousa Ribeiro. -- Rio de Janeiro, 2018.
44 f.

Orientador: Maria da Silva.
Trabalho de conclusão de curso (graduação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2018.

1. Assunto 1. 2. Assunto 2. I. Silva, Maria da, orient. II. Titulo.

Elaborado pelo Sistema de Geração Automática da UFRJ com os dados fornecidos pelo(a) autor(a).

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

Trabalho de conclusão de curso de graduação apresentado ao Instituto de Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Cláudio Esperança
Titulação (Instituição)

Nome do Professor1
Titulação (Instituição)

Nome do Professor2
Titulação (Instituição)

"The display is the computer."

Jen-Hsun Huang

RESUMO

Neste trabalho é descrito elementos básicos para um esquema de modelagem baseado em física de objetos rígidos e deformáveis adequado para aplicações interativas que é simples, rápida e bastante estável. Estes corpos são representados por um grupo de partículas que devem satisfazer um conjunto de restrições lineares por um método de relaxamento e a simulação de seu movimento é usado um esquema de integração Verlet. A detecção das colisões é tratada em duas fases: de *Broad Phase* responsável por reduzir o número de candidatos à colisão com estruturas de divisão espacial e uso de testes rápidos e baratos; de *Narrow Phase* responsável por usar algoritmos mais sofisticados para detectar colisão como o Teorema do Eixo Separador (Separating Axis Theorem - SAT) e algoritmo de Gilbert, Johnson e Keerthi (1988) (GJK). Para a resposta a colisão é usado as técnicas descritas por Jakobsen (2001) pela projeção das posições das partículas envolvidas através do Vetor de Translação Mínimo (minimum translation vector - MTV). Diferente das abordagens tradicionais, a simulação física é obtida sem se computar explicitamente matrizes de orientação, torques ou tensores de inércia.

Palavras-chave: Computação gráfica; Animação Física; Detecção de Colisão; Resposta a Colisão; Corpos rígidos e deformáveis; web.

ABSTRACT

Abstract in english. The text should be typed in a single paragraph with **single spacing** and contain between 150 and 500 words. Use the third person singular, the verbs in the active voice and avoid the use of symbols and contractions that are not of current use. The keywords must appear right below the abstract, preceded by the expression **Keywords:**, separated by a semicolon (;) and ending with a period. They must be written with the initials in lowercase, with the exception of proper nouns and scientific names.

Keywords: artificial intelligence; cryptography; data mining; Sociedade Brasileira de Computação; neural network.

SUMÁRIO

1 INTRODUÇÃO

A evolução da capacidade computacional e das placas gráficas nas últimas décadas permitiu o desenvolvimento de simulações visuais cada vez mais realistas. Dentro desse contexto, a **Computação Gráfica** surge como uma área fundamental da ciência da computação responsável por estudar técnicas e algoritmos para gerar, manipular e representar imagens digitais de maneira eficiente e visualmente convincente. Essa área abrange desde o simples desenho de primitivas geométricas até a renderização de ambientes tridimensionais complexos, sendo amplamente utilizada em aplicações de engenharia, design, entretenimento e educação.

Segundo Azevedo (2003), a computação gráfica é matemática e arte. Esta ferramenta proporciona um maior poder de abstração, ajudando na criação de imagens complexas e em muitos casos não imaginadas. A computação gráfica pode ser encarada como uma ferramenta não convencional que permite ao artista transcender das técnicas tradicionais de desenho ou modelagem.

Em paralelo, o avanço dos **programas interativos**, especialmente jogos eletrônicos e simulações físicas em tempo real, intensificou a necessidade de técnicas capazes de combinar realismo visual com desempenho computacional. De acordo com Möller, Haines e Hoffman (2018), renderização em tempo real refere-se à criação rápida de imagens no computador. É a área mais interativa da computação gráfica: uma imagem é exibida, o usuário reage, e essa reação influencia as próximas imagens a serem geradas.

Esse ciclo de interação deve ocorrer a uma taxa suficientemente alta para manter a sensação de fluidez, medida em quadros por segundo (FPS). A partir de aproximadamente 6 FPS já é possível perceber interatividade, e taxas mais elevadas tornam a experiência imersiva.

Tais programas exigem que o computador responda dinamicamente às ações do usuário, atualizando continuamente o estado do mundo virtual de acordo com as leis físicas e as interações entre objetos. Para atingir essa responsividade, é essencial empregar algoritmos eficientes de *detecção de colisões*, *resposta física* e *atualização de estados*.

1.1 CONTEXTUALIZAÇÃO DO PROBLEMA

Em animações tradicionais baseadas em quadros-chave (*keyframe animation*), o movimento dos objetos é previamente definido, o que limita a capacidade de gerar comportamentos emergentes e interações físicas naturais. Para alcançar resultados mais dinâmicos, recorre-se à **animação baseada em física**, onde forças, restrições e colisões determinam o movimento de forma simulada. Contudo, implementar esse tipo de sistema envolve desafios significativos: é necessário equilibrar precisão física e desempenho computacional,

especialmente porque a detecção de colisões — responsável por identificar interpenetrações e manter a consistência física da simulação — tende a ser a etapa mais custosa. Em cada instante de tempo, o sistema deve atualizar velocidades, forças e demais grandezas físicas, o que, em cenários com muitos objetos, dificulta a execução em tempo real.

Para lidar com essa complexidade, diversas técnicas são empregadas na literatura, como a **detecção de colisões em múltiplas fases** (Broad Phase e Narrow Phase), o uso de **estruturas espaciais otimizadas** (como grades uniformes) e estratégias de **processamento paralelo**. Além disso, diferentes abordagens de resposta física são utilizadas, como o método do impulso, o método da penalização e métodos baseados em restrições.

1.2 MÉTODOS DINÂMICOS NA SIMULAÇÃO FÍSICA

A literatura apresenta diversas abordagens para modelar o movimento de corpos rígidos e deformáveis. Embora este trabalho se baseie no método simplificado de Jakobsen, é importante contextualizar outras categorias amplamente utilizadas em motores físicos modernos: métodos *impulse-based*, métodos *penalty-based* e métodos *constraint-based* via multiplicadores de Lagrange.

Método Método do Impulso

Os métodos *Método do Impulso* tratam colisões aplicando impulsos instantâneos que alteram diretamente as velocidades dos corpos para preservar o momento linear e angular. Essa abordagem é amplamente descrita em **??**) e utilizada em motores como Havok e Bullet. O impulso é calculado em função da velocidade relativa no ponto de contato, resultando em um método eficiente para simulações em tempo real.

Método Metodo de Penalidades

Nos métodos *Metodo de Penalidades*, colisões são tratadas como interpenetrações que geram forças de repulsão proporcionais à profundidade de penetração. Essas forças geralmente seguem modelos de mola e amortecimento, como apresentado por **??**). Trata-se de um método simples, porém sensível à escolha dos parâmetros de rigidez, podendo causar instabilidade numérica.

Método Constraint-Based com Multiplicadores de Lagrange

Os métodos baseados em restrições formulam os contatos como equações que devem ser satisfeitas exatamente. São resolvidos usando multiplicadores de Lagrange, como descrito em **??**) e **??**). Essa abordagem é robusta e adequada para sistemas complexos, mas exige a solução de sistemas lineares, tornando sua aplicação onerosa em plataformas Web.

1.3 MOTORES FÍSICOS E EVOLUÇÃO HISTÓRICA

Motores físicos comerciais e de código aberto moldaram a evolução das técnicas de simulação em tempo real. O **Havok** popularizou o uso profissional de física em jogos AAA, oferecendo um sistema robusto de colisões e restrições. O **NVIDIA PhysX** introduziu aceleração por GPU, permitindo simulações mais ricas. Motores como **Bullet Physics** e **Box2D**, ambos de código aberto, democratizaram o acesso a ferramentas de alta qualidade, tornando-se amplamente adotados em pesquisas, jogos independentes e aplicações embarcadas.

Essas ferramentas influenciaram profundamente metodologias modernas, e muitos dos algoritmos estudados neste trabalho derivam ou são inspirados em conceitos consolidados por esses motores.

1.4 APLICAÇÕES WEB E MOTIVAÇÃO TECNOLÓGICA

O desenvolvimento moderno de aplicações interativas na Web se beneficia da combinação de **WebGL**, **JavaScript** e **Web Workers**. WebGL proporciona renderização acelerada por GPU diretamente no navegador, enquanto JavaScript garante ampla acessibilidade e rápida prototipação. A utilização de Web Workers permite distribuir a simulação física em paralelo, evitando bloqueios na *main thread* e mantendo o desempenho da renderização.

Esse ecossistema torna a plataforma Web um ambiente cada vez mais relevante para jogos, simulações científicas e aplicações educacionais, motivando o foco deste trabalho em uma solução física simplificada e otimizada para navegadores.

1.5 JUSTIFICATIVA

O estudo e a implementação de um sistema de animação física simplificada representam uma oportunidade de unir teoria e prática em Computação Gráfica e Física Computacional. Além de contribuir para a compreensão de conceitos fundamentais, um motor físico otimizado possui aplicações diretas em jogos digitais, visualização científica, engenharia e realidade virtual.

Jakobsen (2001) propôs um esquema simplificado de simulação física, capaz de modelar objetos rígidos e deformáveis sem calcular torques ou tensores de inércia explicitamente. Seu método combina um integrador de Verlet, manutenção iterativa de restrições, resposta a colisões por projeção e uso de aproximações eficientes.

Entretanto, o método omite diversos detalhes importantes, como estratégias de detecção de colisões, tratamento de um grande número de restrições e mecanismos de otimização para múltiplos objetos — lacunas que este trabalho busca explorar.

1.6 OBJETIVOS

Neste trabalho, o objetivo é desenvolver um protótipo inspirado no método de Jakobsen, apresentando soluções para detecção e resposta a colisões com foco em aplicações Web. São metas específicas:

- Revisar os principais conceitos de animação baseada em física e integração numérica;
- Implementar algoritmos de detecção de colisão
- Desenvolver uma simulação física baseada em partículas e restrições utilizando o método de Jakobsen;
- Aplicar técnicas de otimização e processamento multi-threaded;
- Avaliar o desempenho e a estabilidade do sistema em diferentes cenários.

1.7 ESTRUTURA DO TRABALHO

O Capítulo 2 apresenta conceitos fundamentais de animação baseada em física. O Capítulo 3 detalha o método simplificado de Jakobsen. O Capítulo 4 discute metodologias de detecção de colisões. O Capítulo 5 aborda o cálculo das fronteiras simplificadas dos objetos. O Capítulo 6 descreve o modelo de resposta a colisões via projeção de posições. O Capítulo 7 apresenta otimizações empregadas no sistema. O Capítulo 8 discute experimentos e comparações com outras bibliotecas. Por fim, o Capítulo 9 traz considerações finais e trabalhos futuros.

2 ANIMAÇÃO BASEADA EM FÍSICA

A animação baseada em física é uma abordagem de geração de movimento que aparenta seguir princípios físicos básicos, mesmo que as equações envolvidas sejam tratadas de forma aproximada ou altamente simplificada. Diferentemente da animação tradicional, na qual o animador define manualmente posições e rotações ao longo do tempo (*keyframe animation*), a animação física permite que o comportamento dos objetos emerja naturalmente das forças, restrições e interações entre os corpos simulados.

Nesse contexto, a animação física simplificada busca um equilíbrio entre precisão e desempenho: modelos matemáticos são utilizados como guia, mas a prioridade está na estabilidade visual e na resposta interativa. Diferentemente da simulação científica, onde precisão e correção numérica são cruciais, na animação para fins gráficos ou interativos **o objetivo não é obter resultados fisicamente corretos, mas sim um comportamento verossímil**. O foco está em transmitir sensação de peso, inércia e colisões de maneira convincente ao usuário, mesmo quando obtidos por heurísticas.

2.1 CONCEITOS E DEFINIÇÕES

O objetivo central da animação baseada em física é resolver numericamente as equações que descrevem o movimento de objetos em um mundo virtual. Tais equações derivam das leis fundamentais da mecânica clássica, formuladas por Isaac Newton.

> **Primeira lei de Newton.** Na ausência de forças externas, um objeto em repouso permanece em repouso e um objeto em movimento continua em movimento com velocidade constante. Apenas forças externas podem alterar o estado de movimento.

> **Segunda lei de Newton.** Para um corpo de massa constante m submetido a uma força $\vec{F}$, o movimento é descrito por:
>
> $$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{x}}{dt^2}. \tag{2.1}$$

> **Terceira lei de Newton.** Para toda força exercida em um corpo existe uma força de igual magnitude e direção oposta exercida no corpo que a gerou.

Integrando a aceleração ao longo do tempo obtêm-se velocidade e posição, que determinam a trajetória dos objetos. Além das forças, a simulação deve também considerar colisões, atrito, restituição e restrições entre corpos (como juntas). Em implementações simplificadas, esses efeitos são aproximados por regras empíricas e técnicas numéricas que priorizam eficiência computacional.

De maneira geral, um ciclo de simulação física engloba:

1. coleta e soma das forças aplicadas (gravidade, vento, atrito etc.);
2. integração temporal das equações de movimento;
3. detecção e resposta a colisões;
4. atualização das posições e posterior renderização.

O método descrito por Jakobsen (2001) reúne um conjunto de técnicas simples, estáveis e eficientes, que permitem obter resultados visualmente satisfatórios mesmo com aproximações físicas significativas. Seu algoritmo iterativo permite aumentar a precisão ao custo de desempenho, ajustando dinamicamente essa relação conforme a necessidade. O sucesso da abordagem se deve à combinação entre o integrador Verlet, a solução de restrições por relaxamento e uma estratégia de resolução de colisões baseada em projeção. A seguir, descrevem-se os principais componentes desse método.

2.2 REPRESENTAÇÃO DE CORPOS RÍGIDOS

Um **corpo rígido** é um objeto cuja forma e volume permanecem invariáveis durante a simulação. Em termos matemáticos, a distância entre quaisquer dois pontos do corpo é constante, independentemente das forças aplicadas. Essa suposição simplifica o problema, permitindo representar o corpo apenas por grandezas globais: posição, orientação e velocidades linear e angular.

A representação matemática de um corpo rígido é dada por:

- **Posição** $\vec{p}$: coordenadas do centro de massa;
- **Orientação** R: matriz de rotação ou quaternion;
- **Velocidade linear** $\vec{v}$: variação temporal da posição;
- **Velocidade angular** $\vec{\omega}$: variação temporal da orientação;
- **Massa** m e **tensor de inércia** I: medidas de resistência à aceleração.

A dinâmica translacional e rotacional é governada pelas equações:

$$m \cdot \frac{d\vec{v}}{dt} = \sum \vec{F}, \tag{2.2}$$

$$I \cdot \frac{d\vec{\omega}}{dt} = \sum \vec{\tau}, \tag{2.3}$$

onde $\sum \vec{F}$ é o somatório das forças externas e $\sum \vec{\tau}$ o somatório dos torques. Em simulações mais simples — como tecidos, cordas ou partículas — a rotação é frequentemente ignorada, reduzindo a complexidade computacional.

2.3 DINÂMICA DE PARTÍCULAS

A **dinâmica de partículas** é uma abordagem na qual o sistema é composto por partículas independentes. Cada partícula possui posição, velocidade e massa, e suas interações são modeladas por forças (como gravidade ou molas) ou por restrições geométricas (como manter distâncias constantes).

O estado de uma particula no instante t é dado por:

$$X(t) = \begin{pmatrix} x(t) \\ v(t) \end{pmatrix}.$$

Seja $F(t)$ a soma das forças que atuam sobre a partícula e m sua massa. O movimento pode ser descrito por:

$$\frac{d}{dt}X(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \dfrac{F(t)}{m} \end{pmatrix}. \tag{2.4}$$

A dinâmica de partículas é amplamente utilizada para simular tecidos, fluidos e efeitos visuais (como fumaça, poeira ou explosões) devido à sua flexibilidade e à capacidade de gerar comportamentos complexos emergentes a partir de regras simples.

3 MÉTODO SIMPLIFICADO DE JAKOBSEN

O método proposto por Jakobsen (2001) apresenta uma abordagem simples e eficiente para simulação física em tempo real, especialmente voltada para jogos e animações interativas. Sua formulação utiliza um modelo baseado em partículas e restrições geométricas, evitando explicitamente a solução de equações diferenciais rígidas e instáveis. Em vez disso, correções iterativas são aplicadas diretamente às posições das partículas, proporcionando estabilidade numérica elevada e comportamento visualmente plausível mesmo sob passos de tempo grandes.

3.1 MÉTODOS NUMÉRICOS EM SIMULAÇÃO

A simulação de movimento depende da solução numérica das equações diferenciais que descrevem a dinâmica dos corpos. Diversos métodos de integração podem ser utilizados, cada um com um equilíbrio distinto entre precisão, estabilidade e custo computacional.

3.1.1 Método de Euler

O método de Euler explícito é o mais simples e intuitivo. Ele atualiza a posição e a velocidade de acordo com a aceleração atual:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \tag{3.1}$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_t \Delta t \tag{3.2}$$

Apesar de sua simplicidade, o método de Euler tende a ser numericamente instável, especialmente em sistemas oscilatórios (como molas), pois o erro de integração cresce rapidamente ao longo do tempo.

3.1.2 Método Semi-implícito de Euler

Uma variação estável do método de Euler consiste em atualizar primeiro a velocidade e depois a posição, utilizando a nova velocidade no cálculo:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \tag{3.3}$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \Delta t \tag{3.4}$$

Essa pequena modificação melhora a conservação de energia e reduz a instabilidade numérica, sendo amplamente utilizada em motores de física em tempo real.

3.1.3 Integração de Verlet

O método de Verlet é uma alternativa amplamente adotada em simulações físicas de partículas. É um método estável, sua velocidade é calculada implicitamente o quê mantém sua posição e velocidade em sincronia, muito popular em simulação de dinâmica molecular.

$$\vec{x}_{t+\Delta t} = 2\vec{x}_t - \vec{x}_{t-\Delta t} + \vec{a}_t \Delta t^2 \tag{3.5}$$

O método de Verlet é estável e eficiente, especialmente em sistemas sujeitos a restrições geométricas além de eliminar a necessidade de armazenar explicitamente a velocidade, utilizando as posições atual e anterior para estimar a nova posição:

3.2 RESTRIÇÕES GEOMÉTRICAS

O sucesso do método de Jakobsen (2001) está na utilização de restrições geométricas simples para compor estruturas complexas. Objetos físicos são representados como conjuntos de **partículas** conectadas por **restrições**, que impõem relações a serem satisfeitas a cada passo de simulação.

3.2.1 Restrição Linear

O caso mais comum é a **restrição linear** (ou restrição de distância), que mantém uma distância fixa d entre duas partículas i e j. Dessa forma o conjunto de partículas devem satisfazer a todo instante uma coleção de inequações unilaterais representadas na forma:

$$|\vec{x_i} - \vec{x_j}| - d = 0 \tag{3.6}$$

Mesmo que as posições das partículas estejam inicialmente corretas, depois de um passo de simulação a distância entre elas pode se tornar inválidas (através de forças externas, por exemplo). Para corrigir sua distância devemos mover (projetar) elas de tal forma que satisfaça 3.6.

Inserir figura

A essência de uma restrição é a projeção. Deve-se encontrar o movimento mínimo que a satisfaça. O efeito de uma restrição linear pode representar conectar duas partículas com uma haste rígida mas também projetar o ponto em um círculo ao redor do ponto de ancoragem.

Algoritmo 1: Restrição Linear

$\vec{\delta} \leftarrow \vec{x}_2 - \vec{x}_1$
$c \leftarrow \frac{\|\vec{\delta}\| - d}{\|\vec{\delta}\|}$
$\vec{x}_1 \leftarrow \vec{x}_1 - \epsilon\,\vec{\delta}\,c$
$\vec{x}_2 \leftarrow \vec{x}_2 + \epsilon\,\vec{\delta}\,c$

O pseudocódigo 1 irá separar ou aproximar as partículas de tal forma que satisfaçam a distancia d. Essa situação é comparável a um sistema de molas interconectadas entre partículas de rigidez que tendem para o infinito ou a uma haste rígida separando as duas partículas. Quando $\epsilon = 0.5$, as partículas se movem proporcionalmente, valores maiores simulam uma haste mais rígida.

3.2.2 Restrição de Revolução

Para permitir que uma partícula gire em torno de um ponto fixo, basta impor uma restrição de distância entre a partícula e um ponto-âncora.

(Inserir figura ilustrativa.)

Uma forma equivalente é definir uma restrição linear com distância $d = 0$ entre duas partículas, fazendo com que uma permaneça colada à outra.

3.2.3 Restrição Angular

Em muitas situações em animação física é desejável que o ângulo formado entre duas partículas esteja restrito a um intervalo válido. Isso pode ser feito de forma simples aplicando uma restrição linear caso a distância entre duas partículas seja menor que um limiar. Ou seja, satisfazer a inequação abaixo

$$|x_2 - x_1| > d$$

Com essa restrição conseguimos relações geométricas para joelhos e cotovelos de uma criatura que não podem dobrar além de ângulo máximo.

A rotina para essa restrição é tão simples quanto usar um condicional junto com o algoritmo de restrição linear.

Algoritmo 2: Restrição Angular

$\vec{\delta} \leftarrow \vec{x_2} - \vec{x_1}$
if $|\vec{\delta}| < d$ **then**
  $c \leftarrow \frac{|\vec{\delta}| - d}{|\vec{\delta}|}$
  $x_1 \leftarrow x_1 - \vec{\delta} * \epsilon * c$
  $x_2 \leftarrow x_2 + \vec{\delta} * \epsilon * c$

Um outro método de restringir os ângulos é satisfazer a restrição de produto interno

$$(x_2 - x_0) \cdot (x_1 - x_0) < \alpha$$

3.3 RESOLVENDO RESTRIÇÕES CONCORRENTES POR RELAXAMENTO

Na prática, uma simulação pode conter muitas restrições de todos os tipos vistos anteriormente. Para satisfazer todas elas devemos resolver todas as inequações sequencialmente, como as restrições entre partículas são interdependentes, não é possível satisfazê-las todas simultaneamente de maneira exata em um único passo.

Jakobsen (2001) propõe um método iterativo de relaxamento, também conhecido como *Gauss-Seidel relaxation*, para resolver as restrições de forma aproximada. É uma abordagem de solução indireta por iteração local que consiste em aplicar pequenas correções de posição para cada par de partículas conectado, repetindo o procedimento diversas vezes até que todas as restrições estejam aproximadamente satisfeitas. Cada iteração contribui para reduzir o erro acumulado, e a convergência ocorre rapidamente mesmo com poucas iterações (geralmente entre 3 e 5).

Algoritmo 3: Satisfazer Restrições

```
Input: n: número de repetições
for i ← 0 to n do
    foreach restrição em restrições do
        SatisfazerRestricao(restrição)
```

Apesar dessa abordagem parecer ingênua, ao resolver todas restrições localmente e repetir, o sistema global do sistema converge para uma configuração que satisfaça todas restrições. Quanto maior o número de iterações mais rápido o sistema irá convergir para solução e também a animação irá parecer mais rígida para o usuário.

Além disso para o algoritmo 1 o valor e ϵ tem o efeito de aumentar o passo local de convergência para solução ideal. Fisicamente pode ser interpretado como um coeficiente de restituição. Pode ser usado para representar quão abrupto as partículas se aproximam ou afastam.

4 DETECÇÃO DE COLISÕES

O mecanismo de detecção de colisão é um componente fundamental para um sistema de simulação baseada em física. É uma etapa essencial em qualquer sistema de simulação física ou animação baseada em partículas. Seu propósito é identificar quando dois ou mais objetos entram em contato, determinar o ponto de interseção e, opcionalmente, calcular as informações necessárias para a resposta física, como o vetor de penetração e a normal de colisão. Em animações físicas simplificadas, a precisão geométrica pode ser relaxada em favor da eficiência computacional, especialmente em aplicações interativas em tempo real.

Neste capítulo introduzimos conceitos relevantes dentro do contexto deste trabalho e discutimos métodos que geralmente são usados para representar objetos, assim como algoritmos para fazer consultas de proximidade e interseção entre objetos.

4.1 POLÍGONOS CONVEXOS

Um objeto geométrico é um conjunto fechado de pontos, limitado e não vazio. É fechado, pois a borda faz parte do objeto e é limitado significa que existe uma esfera de raio finito que limita o objeto. Assim, por exemplo, um plano é fechado mas não limitado.

(INSERIR IMAGEM FORMA CONVEXA X NAO CONVEXA)

Uma forma é considerada convexa se, para qualquer linha desenhada através da forma, essa linha cruzar apenas duas vezes. Formas não convexas podem ser representadas por uma combinação de formas convexas.

4.2 TEOREMA DO EIXO SEPARADOR (SAT)

O Teorema do Eixo Separador (do inglês Separating Axis Theorem, SAT) é um dos algoritmos mais utilizados para detecção de colisões entre polígonos convexos. O algoritmo também pode ser usado para encontrar o vetor de penetração mínimo que é útil para simulação de física e uma série de outras aplicações.

O SAT é um algoritmo genérico rápido que pode remover a necessidade de ter código de detecção de colisão para cada par tipo de forma, reduzindo assim o código e a manutenção. Ele se baseia no teorema geométrico que afirma:

Teorema 1. *Dois polígonos convexos A e B não se intersectam se, e somente se, existir um eixo (reta) sobre o qual as projeções de A e B não se sobrepõem.*

Um plano de separação (PS) é um plano que está localizado entre dois objetos convexos, separando-os. Eventualmente, A está situado no lado positivo e B no lado negativo

ou vice-versa. Matematicamente, um PS é definido por $H(v, \delta)$, onde v é chamado de eixo de separação (ES).

Se v é um eixo de separação, então existe um escalar δ tal que o plano de separação possa ser definido.

INSERIR IMAGEM POLIGONO SEPARADO E POLIGONO EM COLISÃO

Em termos computacionais, o SAT testa um conjunto finito de eixos candidatos — tipicamente os vetores normais às arestas de ambos os polígonos — e verifica se existe algum eixo que separa os dois conjuntos.

4.2.1 Projeção sobre um eixo

Dado um eixo unitário $\hat{n}$ e um conjunto de vértices $\{v_i\}$, a projeção escalar do polígono sobre $\hat{n}$ é dada pelos extremos:

$$\min_A = \min_i(\hat{n} \cdot v_i^A) \tag{4.1}$$

$$\max_A = \max_i(\hat{n} \cdot v_i^A) \tag{4.2}$$

De forma análoga, para o segundo polígono B:

$$\min_B = \min_j(\hat{n} \cdot v_j^B) \tag{4.3}$$

$$\max_B = \max_j(\hat{n} \cdot v_j^B) \tag{4.4}$$

Se os intervalos $[\min_A, \max_A]$ e $[\min_B, \max_B]$ não se sobrepõem em algum eixo $\hat{n}$, então os polígonos não colidem.

4.2.2 Teste de interseção booleana

O SAT deve testar todos os eixos candidatos a separação para determinar se há ou não sobreposição. Devido a isso ele não é muito prático em ambientes 3D. No entanto, o teorema 1 nos garante que o primeiro eixo onde as projeções não estão sobrepostas, o algoritmo pode sair imediatamente determinando que as formas não estão se cruzando.

O teste de colisão entre dois polígonos convexos A e B via SAT é implementado conforme o seguinte pseudocódigo:

Algoritmo 4: Teste booleano de colisão via SAT

Input: Polígonos convexos A e B
Output: Verdadeiro se houver interseção
foreach *aresta e de A e B* **do**
  $\hat{n} \leftarrow$ normal unitária de e
  $p_i \leftarrow$ Projetar A no eixo n
  $p_j \leftarrow$ Projetar B no eixo n
  if p_i *sobrepoe* p_j **then**
    return *False*
return *True*

Este procedimento tem complexidade linear no número total de arestas e fornece um resultado booleano eficiente para corpos convexos.

4.2.3 Cálculo do Vetor de Translação Mínima (MTV)

Além de determinar se há colisão, muitas vezes é necessário calcular o *vetor de translação mínima* (*Minimum Translation Vector*, MTV), que indica o menor deslocamento necessário para separar dois polígonos sobrepostos.

Para o SAT a direção é equivalente ao eixo de separação e a penetração (magnitude do MTV) é o tamanho da menor projeção do polígono com o eixo de separação. Podemos adaptar nosso pseudocódigo como

Algoritmo 5: SAT completo

Input: Polígonos convexos A e B
Output: separacao v e penetracao δ ou falso
$v \leftarrow$ Vector.Zero
$\delta \leftarrow 0$
foreach *aresta e de A e B* **do**
  $\hat{n} \leftarrow$ unitária de e
  p1 $\leftarrow$ Projetar A no eixo $\hat{n}$
  p2 $\leftarrow$ Projetar B no eixo $\hat{n}$
  $\delta \leftarrow$ projeção p1 com p2
  if $\delta \leq$ *TOLERANCIA* **then**
    return *False*
  else
    $\delta \leftarrow \delta$
    $v \leftarrow$ eixo
return v, δ

4.3 ALGORITMO GILBERT–JOHNSON–KEERTHI (GJK)

O algoritmo de distância Gilbert-Johnson-Keerthi (GJK) é um método de determinar a distância mínima entre dois conjuntos convexos, publicado pela primeira vez por Gilbert,

Johnson e Keerthi (1988). Ao contrário de muitos outros algoritmos de distância, não requer que os dados de geometria sejam armazenados em qualquer formato específico, mas depende apenas de uma função de suporte para gerar iterativamente Simplex mais próximas da resposta correta usando a diferença de Minkowski.

O GJK, como a SAT, só opera em formas convexas. GJK é um método iterativo, mas converge muito rápido. É uma alternativa melhor para o SAT para ambientes 3D devido ao número de eixos que o SAT deve testar.

4.3.1 Soma de Minkowski

O algoritmo GJK depende muito de um conceito chamado soma de Minkowski de dois objetos convexos A e B é definida por:

$$A + B = \{\vec{x} + \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{4.5}$$

O objeto A + B é o conjunto de pontos obtido por um processo de varredura que translada o centro de massa de B para cada ponto de A, ou seja, faz-se uma cópia do objeto B centrado em cada ponto de A

(INSERIR FIGURA SOMA DE MINKOWSKI)

Uma propriedade muito útil da soma de Minkowski é o fato de que a soma de dois objetos convexos é um objeto convexo. O algoritmo GJK se beneficia dessas propriedades usando uma operação informalmente chamada de *diferença de Minkowski*

$$A - B = \{\vec{x} - \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{4.6}$$

Essa operação continua sendo a soma de Minkowski (a soma da diferença). Mas neste trabalho usaremos esse termo para referir a essa operação quando necessário.

Executar essa operação exige $|A| * |B| * 2$ subtrações. Isso é significativo porque uma forma é composta de um número infinito de pontos. Uma vez que ambas as formas são convexas e definidas por vértices mais externos só precisamos realizar esta operação nos vértices. A grande coisa sobre GJK é que você realmente não precisa calcular a diferença de Minkowski para todos os vértices.

A diferença de Minkowski pode ser pensado como um processo de varredura que calcula o vetor distancia para cada ponto de B em A. Dessa forma, caso a distância entre dois pontos seja zero (interseção), podemos confirmar a colisão entre os dois objetos. Esse vetor também coincide com a origem, logo elaboramos nosso próximo teorema:

Teorema 2. *Os conjuntos A e B colidem se, e somente se, o ponto de origem $(0, 0)$ estiver contido em $A \ominus B$.*

Dessa forma, não queremos calcular a diferença de Minkowski. Em vez disso, queremos apenas saber se a Diferença de Minkowski contém ou não a origem. Se isso acontecer, então sabemos que as formas estão se cruzando, se não o faz, então elas não são.

Isso é feito construindo iterativamente um polígono dentro da diferença de Minkowski que tenta incluir a origem. Se o polígono que construímos contém a origem, então podemos dizer que a Diferença de Minkowski contém a origem, e também que há interseção entre os dois objetos. Este polígono que queremos construir deve ser da forma mais elementar possível, no caso de 2D um triângulo, no caso de 3D um poliedro, por isso é chamado de Simplex.

4.3.2 Função de Suporte

O algoritmo GJK começa com um simplex inicial e o refina iterativamente adicionando pontos encontrados usando a função de suporte em uma direção que aponta para a origem. Esse processo continua até que a origem seja encontrada dentro do simplex, indicando uma colisão, ou até que se determine que a origem não está contida, o que significa que não há colisão.

A função de suporte deve retornar o ponto mais distante em uma direção dentro da Diferença de Minkowski. Isso cria um Simplex que contém uma área máxima, aumentando, portanto, a chance de que o algoritmo termine rapidamente. Além disso, podemos usar o fato de que todos os pontos retornados desta forma estão na borda da Diferença Minkowski e, portanto, se não pudermos adicionar um ponto além da origem ao longo de alguma direção, sabemos que a Diferença de Minkowski não contém a origem. Isso aumenta as chances de o algoritmo sair rapidamente em casos de não interseção.

4.3.3 Construção iterativa do simplex

O GJK constrói iterativamente um *simplex* (ponto, segmento, triângulo ou tetraedro, dependendo da dimensão) que aproxima a origem dentro de $A \ominus B$.

Algoritmo 6: GJK

Input: Polígonos convexos A e B

Output: Veradadeiro se ocorreu colisão

Escolhe-se uma direção $\vec{d}$;

Obtém-se um novo ponto $p = \text{support}(A \ominus B, \vec{d})$

Se o novo ponto não avança em direção à origem ($p \cdot \vec{d} < 0$), não há colisão

Caso contrário, atualiza-se o simplex e redefine-se $\vec{d}$ em direção à origem.

5 O FECHO CONVEXO

O Fecho Convexo (FC) de um conjunto é definido como a menor região convexa que contenha todos os pontos (também chamado como volume delimitador). Achar o fecho convexo de um conjunto de pontos, por exemplo, é um problema que aparece quando queremos organizar o conjunto, agrupar os pontos em uma região simples.

Em sistemas de simulação física e detecção de colisões, a representação geométrica dos objetos tem impacto direto na eficiência dos cálculos. Formas complexas, compostas por muitos vértices e superfícies não convexas, tornam os testes geométricos computacionalmente caros.

Para contornar esse problema, é comum empregar aproximações convexas ou volumes delimitadores — denominados *fechos convexos* — que permitem realizar testes mais rápidos sem comprometer significativamente a precisão da simulação.

INSERIR FIGURA DO FECHO CONVEXO DE OBJETOS EM INTERSECÇÃO E SEM INTERSECÇÃO

A ideia é que formas mais simples (como caixas e esferas) tenham testes de sobreposição mais baratos do que os objetos complexos que eles delimitam. O uso do FC permite testes rápidos de rejeição de sobreposição, pois só é necessário testar a geometria complexa delimitada quando a consulta inicial de sobreposição para os volumes delimitadores resulta em um resultado positivo. Em algumas aplicações, o próprio teste de interseção dos volumes delimitadores serve como prova suficiente de colisão.

Segundo Möller, Haines e Hoffman (2018), nem todos os objetos geométricos servem como volumes delimitadores eficazes. As propriedades desejáveis para volumes delimitadores incluem:

- Testes de interseção de baixo custo
- Ajuste preciso
- Cálculo econômico
- Fácil de girar e transformar
- Consome pouca memória

INSERIR IMAGENS COM TIPOS DIFERENTES FORMAS DE FECHO CONVEXO

A ideia principal por trás dos volumes delimitadores é preceder testes geométricos complexos com testes menos dispendiosos que permitem que o teste seja interrompido precocemente. Para suportar testes de sobreposição de baixo custo, o volume delimitador

deve ter uma forma geométrica simples. Ao mesmo tempo, para tornar o teste de corte antecipado o mais eficaz possível, o volume delimitador também deve ser o mais ajustado possível, resultando em um trade-off entre o ajuste e o custo do teste de interseção.

5.1 CAIXA DELIMITADORA ALINHADA AO EIXO COORDENADO (AABB)

A caixa delimitadora mínima para um conjunto de pontos em N dimensões é a caixa com a menor medida (área, volume, ou hipervolume em dimensões superiores) possível que englobe todos os pontos. A caixa delimitadora mínima de um conjunto de pontos é a mesma que a caixa delimitadora mínima de seu FC, um fato que pode ser usado de forma heurística para acelerar a computação.

A caixa delimitadora alinhada aos eixos (em inglês "axis-aligned bounding box", ou AABB) é um dos volumes delimitadores mais comuns. É um paralelepípedo (ou retângulo em 2D) cujas faces são paralelas aos eixos de coordenadas.

Algoritmo 7: Teste de Intersecção AABB

Input: Polígonos convexos A e B

Output: Verdadeiro se estão colidindo

if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  return *False*

if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  return *False*

if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  return *False*

return *True*

5.2 CAIXA ORIENTADA (OBB)

Uma **Oriented Bounding Box** (OBB) é uma caixa retangular que pode estar rotacionada em relação aos eixos globais. Ela é definida por um ponto central c, um conjunto de vetores ortogonais $\hat{u}_i$ representando a orientação, e comprimentos semi-extensões e_i em cada direção:

$$OBB = \left\{ c + \sum_{i=1}^{3} \alpha_i \hat{u}_i \;\middle|\; -e_i \leq \alpha_i \leq e_i \right\} \tag{5.1}$$

OBBs geralmente fornecem um ajuste mais justo em torno de objetos alongados ou rotacionados, reduzindo falsos positivos de colisão. O custo computacional é maior que o de AABBs, mas compensado em simulações com poucos objetos ou com formas muito irregulares.

5.3 ESFERAS E ELIPSOIDES

As **esferas** são os volumes mais simples de todos, definidas apenas por um centro c e um raio r:

$$\text{Sphere} = \{x \in \mathbb{R}^3 \mid \|x - c\| \leq r\} \tag{5.2}$$

Elas permitem testes de colisão extremamente rápidos, mas representam mal objetos com proporções muito distintas. Por esse motivo, esferas e elipsoides são frequentemente empregadas apenas em fases preliminares de detecção, ou como volumes intermediários em hierarquias de limitação (Bounding Volume Hierarchies, BVH).

5.4 QUICKHULL

Quickhull é um algoritmo incremental para Fecho Convexo de um conjunto finito de pontos de qualquer dimensão. Ele usa uma abordagem de divisão e conquista semelhante à do quicksort, da qual seu nome deriva (BARBER; DOBKIN; HUHDANPAA, 1996).

O Quickhull parte de um conjunto de pontos S e constrói o polígono (ou poliedro) convexo que os contém. O processo para 2 dimensões pode ser descrito em linhas gerais da seguinte forma:

Algoritmo 8: Quickhull 2D

Input: Polígono Convexo

Output: Lista dos vértices que representam o fecho convexo

1. Encontre os pontos com coordenadas mínimas e máximas x, pois estes sempre farão parte do casco convexo.
2. Use a linha formada pelos dois pontos para dividir o conjunto em dois subconjuntos de pontos, que serão processados de forma recursiva.
3. Determine o ponto acima da linha com a distância máxima da linha. Este ponto forma um triângulo com os dois pontos na linha.
4. Os pontos dentro desse triângulo não podem ser parte do casco convexo e, portanto, podem ser ignorados nos próximos passos.
5. Repita recursivamente os dois passos anteriores nas duas linhas formadas pelos dois novos lados do triângulo.
6. Repete-se o processo recursivamente para as novas regiões formadas, descartando pontos interiores
7. O processo termina quando todos os subconjuntos estão vazios.

O Quickhull apresenta, em média, complexidade $O(n \log n)$ em duas dimensões, embora em casos degenerados possa chegar a $O(n^2)$. Em três dimensões, o algoritmo é adaptado para construir uma casca poliedral usando o mesmo princípio recursivo, consulte Barber, Dobkin e Huhdanpaa (1996) para melhores definições.

6 RESPOSTA A COLISÃO

A resposta a colisões é uma etapa fundamental em qualquer sistema de simulação física interativa. Após detectar que dois corpos estão penetrando um ao outro, é necessário aplicar um conjunto de correções que restaurem a plausibilidade física do movimento sem introduzir instabilidades. Neste capítulo apresentamos os princípios clássicos, as formulações modernas e a relação direta entre métodos geométricos da fase de detecção, tais como o vetor de translação mínima (MTV), e métodos baseados em partículas e restrições, como o modelo proposto por Jakobsen (2001).

6.1 CONTEXTO HISTÓRICO E MOTIVAÇÃO

Os primeiros motores de física utilizados em gráficos interativos nos anos 1990 empregavam modelos altamente simplificados baseados em impulsos (impulse-based methods), com foco especial em simulações rígidas newtonianas. Embora fisicamente corretos, esses métodos eram computacionalmente caros e exigiam sistemas robustos para resolver forças, torques e empilhamentos estáveis em tempo real.

Já os chamados métodos baseados em penalidades tratam o contato inserindo molas nos pontos de penetração. Embora seja muito simples de implementar, esse método apresenta diversas desvantagens sérias. Por exemplo, é difícil escolher constantes de mola adequadas de forma que, por um lado, os objetos não penetrem demais e, por outro lado, o sistema resultante não se torne instável.

Jakobsen (2001) apresentou um método de simulação baseado em projeção de posição (Position Based Dynamics). Em vez de trabalhar com forças e integrações diferenciais sobre acelerações, os vértices que fazem parte da colisão são simplesmente projetados para fora do obstáculo. Por projeção, em termos gerais, entendemos mover o ponto o mínimo possível até que ele esteja livre do obstáculo.

6.2 FUNDAMENTOS DA RESPOSTA A COLISÃO

A resposta a colisões consiste, essencialmente, em duas operações principais:

1. **Correção de Posição**: remover a interpenetração entre dois corpos.
2. **Correção de Velocidade**: ajustar ou anular componentes da velocidade que provoquem novo contato imediato.

Embora os motores tradicionais realizem ambas as etapas, muitos sistemas baseados em PBD focam totalmente na correção de posições, derivando velocidades implicitamente a partir da diferença entre posições sucessivas.

6.3 PROJEÇÃO DA POSIÇÃO PELO MÉTODO JAKOBSEN

A resposta à uma colisão é composta de dois passos. O primeiro passo consiste em separar os elementos dos objetos (vértice, aresta ou face) que estão se intersectando. Este passo é puramente geométrico, já que é executado movendo as partículas na geometria da colisão.

O segundo passo é um processo de relaxamento iterativo no qual os elementos dos objetos que estão envolvidos na colisão encontram as suas posições apropriadas usando as restrições como suporte.

Se dois objetos convexos A e B colidem o processo de separação requer três parâmetros: os pontos p_A de contato de A, os pontos p_B de contato de B e o ponto q que a simulação assume onde os dois objetos estão em contato chamado de ponto de projeção.

O ponto de projeção coincide com o plano de separação $H(\vec{v}, \delta)$ nas coordenadas locais do objeto, dessa forma como regra geral basta mover os pontos de p_A em direção $H(\vec{v}, \delta)$. Considere dois casos de colisão:

6.4 ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA)

Para realizar a separação de dois objetos usando o algoritmo SAT basta calcularmos o MTV como visto na seção 4.2. Já para o GJK é preciso fazer um segundo passo, uma extensão do algoritmo que nos permite encontrar a normal correta e profundidade das colisões.

O Algoritmo de Expansão de Politopos (do inglês Expanding Polytope Algorithm, EPA) cria um polítopo (ou polígono) dentro da Diferença de Minkowski e iterativamente expandi-lo até atingirmos a borda da Diferença de Minkowski. EPA executa essa tarefa utilizando a mesma função de suporte utilizada nos demais algoritmos e a mesma noção de um simplex.

Este algoritmo é uma extensão porque sua entrada é o Simplex final do GJK que contém a origem e encontra o MTV. A distância entre o ponto mais próximo com a origem é a profundidade de penetração (δ). Além disso, o vetor normal para o ponto mais próximo é a direção de separação (ponto de contato). A solução ingênua é usar o normal da face mais próxima da origem, porém um simplex não precisa conter nenhuma das faces do polígono original, o quê pode acabar com uma normal incorreta.

O algoritmo expande o Simplex adicionando vértices a ele até encontrarmos a normal mais próxima de uma face que está no polígono original.

Algoritmo 9: EPA

Input: Simplex

Output: separation v, penetration δ

for $i \leftarrow 0$ **to** $i < MAX\_ITERATION$ **do**
    e ← Encontrar aresta mais próxima a origem
    p ← Calcular novo ponto de suporte na direção da normal de e
    $\delta \leftarrow p \cdot normal(e)$
    if $|\delta - length(e)| < TOLERANCE$ **then**
        return *normal(e), δ*
    Adicionar ponto ao simplex

É importante limitar o número de iterações para evitar que a rotina entre em loop infinito em casos degenerados, como esse algoritmo converge rapidamente uma constante igual a 30 é um bom limite superior. Matematicamente a distância deve ser igual a zero, mas por conta da artimética de ponto flutuante, uma tolerância pequena deve ser aceita, como 10^{-3}.

7 OTIMIZAÇÕES

A eficiência computacional é um dos fatores determinantes para o desempenho de um motor de física em tempo real. Em jogos, animações interativas, simulações físicas e aplicações gráficas, a demanda por atualizações contínuas e a necessidade de operar dentro de limites rigorosos de tempo tornam indispensável o uso de técnicas de otimização em todos os estágios do pipeline de simulação.

Como qualquer objeto pode potencialmente colidir com qualquer outro objeto, uma simulação com n objetos requer $(n-1)+(n-2)+\cdots+1 = n(n-1)/2 = O(n^2)$ testes de pares, no pior caso. Devido à complexidade de tempo quadrática, testar ingenuamente cada par de objetos para verificar colisões rapidamente se torna muito caro, mesmo para valores moderados de n.

Reduzir o custo associado ao teste de pares afetará o tempo de execução apenas linearmente. Para realmente acelerar o processo, o número de pares testados deve ser reduzido. Essa redução é realizada separando o tratamento de colisões de múltiplos objetos em duas fases: Narrow Phase e Broad Phase.

Neste capítulo descrevemos as principais estratégias empregadas na otimização da detecção e resolução de colisões, com foco na divisão entre *Broad Phase* e *Narrow Phase*, no uso de estruturas espaciais, na adoção de passos de simulação fixos e em técnicas de paralelização por múltiplos núcleos de processamento.

7.1 BROAD PHASE

A fase de *Broad Phase* tem como objetivo eliminar pares de objetos que certamente não estão colidindo, reduzindo o conjunto de pares candidatos antes da execução de testes mais complexos.

Segundo Ericson (2004), o primeiro princípio da otimização é que nada é mais rápido do que não ter que realizar uma tarefa. Dessa forma, as melhores otimizações para acelerar uma rotina giram em torno de reduzir o trabalho ao mínimo possível o mais cedo possível.

A Broad Phase identifica grupos menores de objetos que podem estar colidindo e rejeita rapidamente aqueles que não estão. Como os objetos só podem atingir coisas que estão próximas a eles, os testes contra objetos distantes podem ser evitados. Isso é feito realizando uma consulta espacial para objetos próximos.

A consulta de colisão mais simples é o problema teste de interseção: responder à pergunta booleana se dois objetos A e B estão se sobrepondo em suas posições e orientações dadas. As consultas de interseção booleanas são rápidas e fáceis de implementar e, portanto, são ideias para Broad Phase.

Algoritmo 10: Broad Phase generalizada

```
Output: Pares de objetos passíveis a colisão
vistos ← {}
pares de contato ← {}
foreach bodyA em bodies do
    candidates ← consultar pares próximos de bodyA
    foreach bodyB do
        if par {bodyA, bodyB} já foi visto then
            ir para próximo par
        marcar par como visto
        if teste de intersecção barata then
            pares de contato ← {bodyA, bodyB}
return pares de contato
```

7.1.1 Grade uniforme

As técnicas de partição do espaço é o processo pelo qual o espaço é subdividido em regiões convexas, chamadas células. Cada célula na partição mantém uma lista de referências a objetos que nela estão (parcialmente) contidos. Como os objetos só podem se interceptar se sobrepuserem à mesma região do espaço, o número de testes de pares de objetos é drasticamente reduzido.

Um esquema muito eficaz de subdivisão espacial consiste em sobrepor um espaço com uma grade regular. Essa grade divide o espaço em várias células de tamanho igual. Cada objeto é então associado às células com as quais se sobrepõem.

INSERIR IMAGEM DE DIVISÃO ESPACIAL EM GRANDE UNIFORME

Devido à uniformidade da grade, acessar uma célula correspondente a uma determinada coordenada é simples e rápido: os valores das coordenadas do mundo são simplesmente divididos pelo tamanho da célula para obter as coordenadas da célula. Dadas as coordenadas de uma célula específica, localizar as células vizinhas também é trivial.

Em termos de desempenho, um dos aspectos mais importantes dos métodos baseados em grade é a escolha de um tamanho de célula apropriado. Existem quatro questões relacionadas ao tamanho da célula que podem prejudicar o desempenho:

1. Se as células forem muito pequenas, um grande número de células precisará ser atualizado.

2. Se os objetos forem pequenos e as células da grade forem grandes, haverá muitos objetos em cada célula.

3. Se os objetos tiverem uma geometria muito complexa isso irá afetar os testes de interseção, eles devem ser divididos em partes menores.

4. É possível que os objetos tenham ambas características anteriores. Sendo necessário outra abordagem como grade hierárquicas.

O ideal é que cada objeto caiba exatamente no tamanho de uma célula.

7.2 NARROW PHASE

Após a Broad Phase reduzir a lista de pares, a *Narrow Phase* realiza testes geométricos precisos. Esta fase utiliza algoritmos complexos como:

- SAT (Separating Axis Theorem) para polígonos/poliedros convexos.
- GJK (Gilbert–Johnson–Keerthi) para formas convexas arbitrárias.
- EPA (Expanding Polytope Algorithm) para obtenção da profundidade de penetração.

A complexidade é reduzida aos pares realmente necessários, tipicamente em número linear no tamanho da cena.

7.3 FÍSICA COM TAMANHO DE PASSO FIXO

A tamanho do passo da simulação física é tradicionalmente atribuída como a variação do tempo que o último quadro demorou para finalizar. Essa abordagem trás um passo variável que será executada mais rápido ou mais lento dependendo do computador do usuário.

Algoritmo 11: Simulação física com passo variável

```
tempo anterior ← agora()
while !sair do
    tempo agora ← agora()
    dt ← tempo agora - tempo anterior
    tempo anterior ← tempo agora
    Física(dt)
    Renderizar()
```

O problema dessa abordagem é que o comportamento da sua simulação física depende do tempo que seu computador leva em cada quadro. O efeito pode ser sutil como sua simulação ter comportamento ligeiramente diferente dependendo da taxa de quadros ou pode ser tão ruim quanto para FPS altos sua simulação explodindo para o infinito e para baixos FPS objetos em movimento atravessando paredes

Essa rotina pode ser reescrita considerando que a simulação é bem comportada apenas se o tempo delta é menor ou igual a algum valor máximo. Avance a simulação em passos fixos de tempo e garanta que ela acompanhe os valores de tempo do render.

Algoritmo 12: Simulação física com passo fixo

```
tempo passado ← agora()
fixed dt ← 1/50
acumulador ← 0
while !sair do
    tempo agora ← agora()
    dt ← tempo agora - tempo anterior
    tempo anterior ← tempo agora
    acumulador ← acumulador + dt
    while acumulador <= fixed dt do
        Física(dt)
        acumulador ← acumulador - fixed dt
    Física(dt)
    Renderizar()
```

Isso garante:

- estabilidade numérica,
- previsibilidade temporal,
- resultados reprodutíveis,
- funcionamento adequado de restrições e colisões.

7.4 SIMULAÇÃO FÍSICA MULTI-THREAD

A maioria dos programas interativos segue uma rotina tradicional single-thread: os inputs do usuário são lidos, enviados para lógica do programa acionando atualizações físicas que são renderizadas na tela.

INSERIR DIAGRAMA

Para um sistema single-thread, a renderização só pode começar depois que a física tiver sido simulada, ou seja, é impossível renderizar antes que a simulação física seja feita.

Nos casos em que uma alta quantidade de cálculo é necessária para a simulação de física, a renderização seria atrasada e a simulação o gargalo, resultando em baixas taxas de quadros e falhas gráficas. O contrário também pode ocorrer: a renderização demorar resulta em um atraso na leitura da entrada do usuário e no processo de simulação física.

Tudo isso é devido a todo o processo que está sendo realizado em um único loop. Um sistema de single-thread seria adequado nos casos em que não é necessário muito cálculo e onde não são renderizados muitos elementos. No entanto, problemas ocorrerão se houver uma carga pesada.

INSERIR NOVO DIAGRAMA

Para resolver esse problema devemos executar a simulação física numa thread separada enquanto o thread principal apenas renderiza os dados de física calculados mais recentes.

Dessa forma, a thread principal não precisa esperar a simulação física terminar para renderizar a configuração mais recente dos objetos. Mesmo em situações em que há uma carga pesada na renderização, o cálculo da física pode ser realizado em paralelo.

8 EXPERIMENTOS

Este capítulo apresenta os experimentos realizados com o objetivo de avaliar o desempenho, a estabilidade e a precisão do sistema desenvolvido. O sistema é baseado no método de integração de Verlet proposto Jakobsen (2001), e em algoritmos clássicos de detecção de colisões, notadamente o *Separating Axis Theorem* (SAT) e o *Gilbert–Johnson–Keerthi* (GJK).

Os experimentos também investigam os impactos das otimizações aplicadas nas etapas de *Broad Phase* (utilizando uma grade espacial uniforme), na *Narrow Phase* (empregando SAT e GJK) e na paralelização do cálculo físico por meio de múltiplas threads. O objetivo é verificar a viabilidade dessas técnicas em um ambiente de execução web, onde o custo computacional e a responsividade são fatores críticos.

8.1 AMBIENTE DE TESTE

Os experimentos foram conduzidos em um computador com as seguintes especificações:

- **Processador:** AMD Ryzen 5 1600X @ 3.3GHz
- **Memória RAM:** 16 GB DDR4
- **Sistema Operacional:** Ubuntu 24.04 LTS
- **Plataforma de execução:** Navegador Firefox 121
- **Implementação:** Typescript + Web Workers (multi-threading), Vuejs e p5js

A escolha do ambiente web teve como propósito demonstrar a aplicabilidade de um motor físico leve em contextos multiplataforma, utilizando exclusivamente tecnologias abertas e acessíveis.

8.2 CONFIGURAÇÃO DOS CENÁRIOS

Três grupos de experimentos foram definidos, cada um com foco em um aspecto distinto do sistema proposto:

8.2.1 Experimento 1 - Integrador de Jakobsen

O primeiro experimento avaliou a estabilidade do método de Verlet em comparação com o integrador de Euler explícito. Foram criados sistemas de partículas conectadas por restrições lineares, representando tecidos e correntes.

INSERIR IMAGENS

Cada sistema foi submetido a diferentes passos de tempo ($\Delta t = 1/30s$, $1/60s$ e $1/120s$) e número de iterações de correção de restrições (de 1 a 10). Observou-se o comportamento visual e a divergência de energia ao longo da simulação.

O integrador de Verlet apresenta maior estabilidade sob altas iterações de restrição, ainda que introduza pequenas imprecisões de posição em sistemas altamente rígidos.

8.2.2 Experimento 2 — Detecção de Colisões Convexas (SAT e GJK)

O segundo experimento teve como objetivo comparar os algoritmos de detecção de colisão *Separating Axis Theorem* (SAT) e *Gilbert–Johnson–Keerthi* (GJK) em termos de precisão e custo computacional.

Foram utilizados objetos convexos de 3 a 8 vértices (em 2D). Cada cenário variou de 2 até 100 objetos móveis, gerando colisões dinâmicas com rotações e translações aleatórias.

Os tempos médios de detecção e a taxa de acertos foram medidos com e sem a utilização de uma etapa de **Broad Phase** baseada em *grade uniforme*.

Resultados esperados:

- O algoritmo SAT demonstrou desempenho satisfatório em colisões bidimensionais com poucos vértices.
- A introdução da *Broad Phase* reduziu significativamente o número de pares testados na *Narrow Phase*, resultando em ganho médio de até 65% em desempenho.

8.2.3 Experimento 3 — Simulação Multi-Threaded

O terceiro experimento avaliou os benefícios do uso de concorrência na simulação física. A implementação utilizou a API *Web Workers* para distribuir a atualização das partículas e as verificações de colisão entre múltiplas threads.

Os testes foram realizados com 1, 2, 4 e 8 threads lógicas, medindo-se:

- O tempo médio de atualização física (em milissegundos);
- A taxa de quadros por segundo (FPS) mantida;
- O ganho relativo de desempenho ($S_p = T_1/T_p$).

Resultados esperados: a paralelização da fase de integração e colisão apresentou ganhos quase lineares até quatro threads, com leve saturação de desempenho a partir de seis threads devido à sobrecarga de comunicação entre processos.

8.3 MÉTRICAS DE AVALIAÇÃO

As seguintes métricas foram utilizadas para quantificar o comportamento do sistema:

- **Tempo médio por quadro (ms):** tempo de execução de uma iteração completa da simulação física;
- **Energia total (E):** estabilidade numérica da simulação;
- **Erro médio de restrição (ε):** precisão das restrições físicas;
- **Taxa de colisões corretas:** proporção de colisões detectadas corretamente em relação ao total esperado;
- **Speedup (S_p):** relação entre o tempo de execução com uma thread e com p threads.

8.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos indicam que o método de Jakobsen apresenta um bom equilíbrio entre estabilidade e simplicidade de implementação, sendo especialmente adequado para simulações de tecidos e cadeias articuladas em tempo real.

Os algoritmos SAT e GJK apresentaram comportamentos complementares: o SAT mostrou-se mais simples e eficiente em 2D, enquanto o GJK foi superior para colisões tridimensionais complexas. A combinação de ambos na *Narrow Phase*, precedida pela otimização em grade uniforme na *Broad Phase*, resultou em ganhos expressivos de desempenho sem perda significativa de precisão.

A utilização de múltiplas threads proporcionou melhorias significativas na taxa de atualização da simulação, especialmente em cenários densos com mais de 100 corpos dinâmicos. O gráfico da Figura ilustra a relação entre número de threads e o ganho de desempenho observado.

INSERIR FIGURA

8.5 LIMITAÇÕES E TRABALHOS FUTUROS

Entre as limitações observadas, destacam-se:

- Dificuldade em lidar com colisões múltiplas simultâneas sem penalização de desempenho;
- Necessidade de decomposição prévia de corpos não convexos;

Como trabalhos futuros, propõe-se:

- Realizar testes em ambientes 3D
- Implementar hierarquias de volumes limitadores (BVH);
- Migrar a execução paralela para WebGPU Compute Shaders, permitindo simulação massiva em GPU.

9 CONCLUSÕES

Os experimentos realizados confirmam a viabilidade de um sistema de animação física simplificada, eficiente e estável, totalmente implementado em ambiente web. A combinação entre o integrador de Jakobsen, as otimizações de detecção de colisão e a execução multi-threaded proporcionou resultados consistentes e visualmente plausíveis em tempo real.

Tais resultados demonstram que é possível construir um motor físico leve e acessível, adequado para aplicações educacionais, jogos independentes e simulações científicas interativas, sem a necessidade de bibliotecas externas ou dependências proprietárias.

REFERÊNCIAS

AZEVEDO, A. C. E. **Computação gráfica - Teoria e prática**. [S.l.]: Editora Campus, Ltda, 2003.

BARBER, C. B.; DOBKIN, D. P.; HUHDANPAA, H. The quickhull algorithm for convex hulls. **ACM Transactions on Mathematical Software (TOMS)**, Acm New York, NY, USA, v. 22, n. 4, p. 469–483, 1996.

ERICSON, C. **Real-time collision detection**. [S.l.]: Crc Press, 2004.

GILBERT, E.; JOHNSON, D.; KEERTHI, S. A fast procedure for computing the distance between complex objects in three-dimensional space. **IEEE Journal on Robotics and Automation**, v. 4, n. 2, p. 193–203, 1988.

JAKOBSEN, T. Advanced character physics. In: **Proceedings of the Game Developer's Conference 2001**. San Jose, CA: Game Developers Conference, 2001. Presented at Game Developer's Conference 2001.

MÖLLER, T.; HAINES, E.; HOFFMAN, N. **Real-time rendering**. 4. ed. [S.l.]: CRC Press, 2018.