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

Este trabalho apresenta o desenvolvimento de um sistema de animação física simplificada para aplicações Web, fundamentado no método de Jakobsen (2001). O objetivo é investigar e demonstrar como técnicas de simulação leve podem produzir movimentos coerentes, estáveis e visualmente naturais, mesmo em ambientes com recursos computacionais limitados, como navegadores modernos. Para isso, são integradas abordagens clássicas de detecção e resposta a colisões, incluindo estratégias de Broad Phase e Narrow Phase, bem como métodos geométricos amplamente utilizados em sistemas interativos. Além disso, o projeto incorpora otimizações estruturais, como subdivisão espacial e processamento multi-threaded, buscando garantir escalabilidade e desempenho em cenários com múltiplos objetos dinâmicos. Os resultados obtidos evidenciam que é possível alcançar simulações eficientes e responsivas mantendo baixo custo computacional, tornando essas técnicas adequadas para jogos, visualizações interativas e aplicações educacionais na Web.

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

ABSTRACT

This work presents the development of a simplified physical animation system for Web applications, based on Jakobsen (2001)'s method. The goal is to investigate and demonstrate how lightweight simulation techniques can produce coherent, stable, and visually natural motion, even in environments with limited computational resources, such as modern browsers. To achieve this, classical approaches to collision detection and response are integrated, including Broad Phase and Narrow Phase strategies, as well as geometric methods widely used in interactive systems. In addition, the project incorporates structural optimizations such as spatial subdivision and multi-threaded processing, aiming to ensure scalability and performance in scenarios with multiple dynamic objects. The results obtained show that it is possible to achieve efficient and responsive simulations while maintaining low computational cost, making these techniques suitable for games, interactive visualizations, and educational applications on the Web.

Keywords: Computer Graphics; Physics Animation; Collision Detection; Collision Response; Rigid and Deformable Bodies; Web.

SUMÁRIO

1 INTRODUÇÃO

O campo da **Computação Gráfica** evoluiu significativamente nas últimas décadas, impulsionado pelo aumento da capacidade computacional e pela especialização do hardware gráfico. Esta área investiga métodos e algoritmos para gerar, manipular e representar imagens digitais de forma eficiente, desempenhando um papel central em aplicações de design, engenharia, entretenimento e visualização científica. Como destaca Azevedo (2003), a computação gráfica combina matemática e arte, oferecendo meios para representar fenômenos complexos e criar imagens inviáveis pelos métodos tradicionais. Ela pode ser encarada, portanto, como uma ferramenta que permite ao artista transcender as limitações das técnicas convencionais 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) num programa interativo, 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 que o usuário não veja imagens individuais, mas sim se sinta imerso em um processo dinâmico. A taxa na qual as imagens são exibidas é medida em quadros por segundo (FPS) ou Hertz (Hz). Para aplicações interativas, requer-se ao menos uma taxa de 6 FPS sendo que taxas mais elevadas tornam a experiência mais mais imersiva.

No contexto das aplicações interativas, destacam-se aquelas que realizam **animação**, isto é, a produção de imagens dinâmicas que representam objetos em movimento. Em geral, essas animações podem ser classificadas em dois grandes grupos: (i) animações *keyframe*, nas quais artistas definem manualmente os quadros-chave e as interpolações; e (ii) animações **baseadas em física**, em que os movimentos emergem da simulação de leis físicas, permitindo comportamentos naturais e interações complexas entre objetos.

O presente trabalho aborda a implementação de **animação física 2D na Web**. A escolha deste tema justifica-se por sua ampla aplicabilidade em jogos digitais, simuladores educacionais e interfaces visuais interativas. A popularidade do problema decorre de sua relevância prática e de seus desafios teóricos, que envolvem tanto a modelagem geométrica dos objetos quanto sua evolução temporal segundo princípios físicos.

Diversos jogos e motores gráficos modernos adotam animações físicas para produzir experiências imersivas e responsivas. Exemplos incluem sistemas de queda, *ragdolls*, tecidos, corpos rígidos e fluidos. Em todos esses casos, a simulação envolve um conjunto de corpos que possuem propriedades físicas — como massa, forma, velocidade e aceleração — e cujas interações são determinadas por colisões, forças e restrições. Uma animação física requer, portanto, a resolução conjunta de dois domínios: a **geometria**, necessária para

detectar e calcular contatos, e a **física**, responsável pela evolução dinâmica dos corpos.

De forma geral, um sistema de animação física segue o seguinte esquema: (i) definição dos corpos e de suas propriedades físicas; (ii) integração numérica para atualizar suas posições e velocidades; (iii) **detecção de colisões**, que identifica se e como os objetos se interceptam; (iv) **resposta às colisões**, que corrige interpenetrações e determina novas velocidades; e (v) renderização dos resultados ao usuário. Por sua natureza iterativa e dependente de múltiplos módulos, trata-se de um problema complexo, sensível ao desempenho e à precisão.

A relevância dessa classe de problemas motivou o desenvolvimento de diversos motores de física que alavancam hardware gráfico para melhorar o desempenho. Entre os quais podemos citar motores físicos comerciais e de código aberto como **Havok** popularizou o uso profissional de física em jogos de grande porte, 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. Tais ferramentas demonstram a importância do tema e como algoritmos otimizados, aliados a hardware moderno, permitem simulações estáveis em tempo real.

Jakobsen propôs durante o desenvolvimento do jogo *Hitman: Codename 47* um esquema simplificado de simulação física que é capaz de modelar corpos rígidos e deformáveis, e tecidos, sem computar explicitamente matrizes de orientação, torques ou tensores de inércia. Seu método combina 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.

Independentemente da complexidade da simulação física, toda aplicação interativa requer um módulo final de **renderização**, responsável por apresentar ao usuário o estado atualizado da cena. No contexto Web, tecnologias como **WebGL** e **Canvas** permitem que essa etapa seja realizada com aceleração gráfica, integrando o fluxo completo de animação física em tempo real.

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 como GJK e SAT
- 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.

O trabalho está organizado da seguinte forma: O Capítulo 2 fundamenta a animação baseada em física e detalha o método de Jakobsen. O Capítulo 3 descreve os algoritmos essenciais para a detecção de colisões (SAT e GJK). O Capítulo 4 aborda as técnicas de resposta a colisão, incluindo projeção de posição e o algoritmo EPA (*Expanding Polytope Algorithm*). O Capítulo 5 foca nas estratégias de otimização para tempo real, cobrindo as fases *Broad Phase* e *Narrow Phase*, volumes delimitadores e processamento *multi-thread*. O Capítulo 6 apresenta a metodologia experimental e a discussão dos resultados. Por fim, o Capítulo 7 resume as contribuições, discute limitações e propõe 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 verossimilhança visual (plausibilidade)**. 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 (Inércia):** 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 (Princípio Fundamental da Dinâmica):** 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 (Ação e Reação):** Para toda força exercida em um corpo existe uma força de igual magnitude e direção oposta exercida no corpo que a gerou.

Em implementações simplificadas, efeitos complexos como atrito, restituição e deformação são aproximados por modelos empíricos. De maneira geral, o ciclo de uma 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.

2.2 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:

$$\frac{d}{dt}X(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \dfrac{F(t)}{m} \end{pmatrix}. \tag{2.2}$$

Esta abordagem é flexível e serve como base para simular sistemas complexos, como tecidos, fluidos e corpos deformáveis, onde o comportamento macroscópico emerge da interação coletiva das partículas.

2.3 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:

$$m \cdot \frac{d\vec{v}}{dt} = \sum \vec{F}, \tag{2.3}$$

$$I \cdot \frac{d\vec{\omega}}{dt} = \sum \vec{\tau}, \tag{2.4}$$

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.

2.4 SIMULAÇÃO DE CORPOS DEFORMÁVEIS

Enquanto a dinâmica de corpos rígidos assume que a distância entre os pontos constituintes do objeto permanece inalterada, a simulação de **corpos deformáveis** lida com objetos que alteram sua forma sob a ação de forças externas. Esta categoria abrange uma vasta gama de fenômenos físicos, desde o comportamento elástico de uma bola de borracha até a dinâmica complexa de tecidos, cabelos e fluidos.

Na mecânica do contínuo, a deformação é descrita pela mudança na configuração métrica do material, gerando tensões internas que tendem a restaurar o objeto ao seu estado de repouso ou dissipar energia na forma de deformação plástica. No contexto da computação gráfica em tempo real, no entanto, modelos discretos simplificados são preferidos em relação aos complexos Métodos de Elementos Finitos (FEM), devido ao seu menor custo computacional.

Sistemas Massa-Mola

A abordagem mais comum para simular deformações em jogos e aplicações interativas é o modelo **Massa-Mola** (*Mass-Spring System*). Neste modelo, um objeto é discretizado em um conjunto de partículas pontuais com massa m, conectadas por molas ideais sem massa.

A força interna exercida por uma mola entre duas partículas i e j é descrita pela Lei de Hooke:

$$\vec{F}_{elastica} = -k_s(|\vec{x}_i - \vec{x}_j| - L_0)\frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|}, \tag{2.5}$$

onde k_s é a constante de rigidez (stiffness), L_0 é o comprimento de repouso da mola e $\vec{x}$ são as posições das partículas. Para garantir a estabilidade numérica e simular a perda de energia, adiciona-se frequentemente um termo de amortecimento (*damping*):

$$\vec{F}_{amortecimento} = -k_d(\vec{v}_i - \vec{v}_j) \cdot \frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|}, \tag{2.6}$$

onde k_d é o coeficiente de amortecimento e $\vec{v}$ são as velocidades.

Estruturação Topológica para Tecidos

Para simular superfícies deformáveis, como tecidos, as partículas são organizadas em uma malha (*grid*). A estabilidade e o comportamento visual do tecido dependem de como essas molas (ou restrições) são conectadas. Uma topologia robusta geralmente emprega três tipos de conexões:

1. **Molas Estruturais (*Structural*):** Conectam partículas vizinhas diretas (horizontal e verticalmente). Elas resistem à tração e compressão básicas, mantendo a integridade da malha.

2. **Molas de Cisalhamento (*Shear*):** Conectam partículas diagonalmente vizinhas. Sua função é impedir que o tecido se distorça ou "deslize"sobre si mesmo, mantendo a estabilidade dos quadriláteros da malha.

3. **Molas de Flexão (*Bend*):** Conectam partículas alternadas (pulando um vizinho). Elas resistem à dobra do tecido, impedindo que ele se comporte como uma malha infinitamente flexível e conferindo-lhe uma certa rigidez à curvatura.

2.5 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.

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:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \tag{2.7}$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_t \Delta t \tag{2.8}$$

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.

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:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \tag{2.9}$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \Delta t \tag{2.10}$$

Essa pequena modificação melhora a conservação de energia e reduz a instabilidade numérica, sendo amplamente adotado em motores como o Box2D.

Integração de Verlet

A Integração de Verlet é a base do método de Jakobsen. É um método de segunda ordem que não armazena explicitamente a velocidade. A nova posição é calculada com base na posição atual, na posição anterior e na aceleração:

$$\vec{x}_{t+\Delta t} = 2\vec{x}_t - \vec{x}_{t-\Delta t} + \vec{a}_t \Delta t^2 \tag{2.11}$$

A velocidade, quando necessária para cálculos de amortecimento ou jogabilidade, é derivada implicitamente:

$$\vec{v}_t \approx \frac{\vec{x}_{t+\Delta t} - \vec{x}_{t-\Delta t}}{2\Delta t} \tag{2.12}$$

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:

2.6 RESTRIÇÕES GEOMÉTRICAS

Jakobsen (2001) utiliza 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.

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:

$$|\vec{x_i} - \vec{x_j}| - d = 0 \tag{2.13}$$

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 2.13.

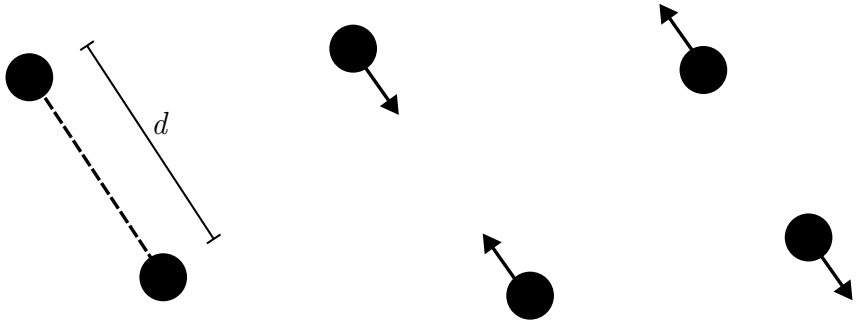

Figura 1 – Correção de posição inválida por projeção. Partículas na posição correta (à esquerda), muito distantes (centro) e muito próximas (à direita)

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. Deve-se encontrar o movimento mínimo que satisfaça a restrição. 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.

Restrição de Revolução

Para permitir que uma partícula gire em torno de um ponto fixo podemos construir um objeto que compartilhe a mesma partícula ou impor uma restrição de distância entre a partícula e um ponto-âncora.

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

Restrição Angular

Para limitar o ângulo de rotação de articulações, pode-se utilizar restrições de distância auxiliares. Para garantir que o ângulo não exceda um limite, impõe-se que a distância entre as extremidades seja maior que um valor d_{min}:

$$|\vec{x}_2 - \vec{x}_1| > d_{min}$$

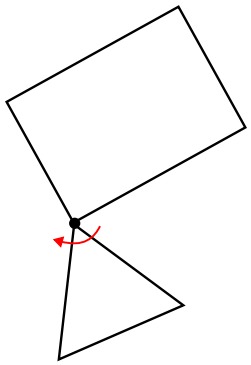

Figura 2 – A restrição de revolução é uma conexão entre dois corpos em um único ponto de ancoragem.

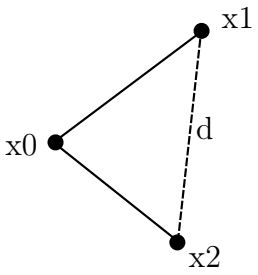

Figura 3 – Duas restrições lineares e uma restrição angular

O algoritmo é análogo ao da restrição linear, porém a correção é aplicada apenas se a condição de desigualdade for violada.

2.7 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).

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.

Algoritmo 2: Satisfazer Restrições

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

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.

Jakobsen 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 DETECÇÃO DE COLISÕES

A detecção de colisões é um componente fundamental em sistemas de simulação física e em animações baseadas em partículas. Esse processo identifica quando dois ou mais objetos entram em contato, determina pontos de interseção e, quando necessário, fornece informações como vetores de penetração e normais que servirão de entrada para a etapa subsequente de resposta física. Em contextos interativos em tempo real, a precisão geométrica absoluta costuma ser sacrificada em favor da eficiência computacional, desde que o comportamento resultante se mantenha visualmente verossímil.

Este capítulo introduz os principais conceitos utilizados no âmbito deste trabalho, descrevendo as representações geométricas mais comuns para objetos convexos e apresentando dois algoritmos amplamente empregados em detecção de colisões: o Teorema do Eixo Separador (SAT) e o algoritmo Gilbert–Johnson–Keerthi (GJK). Ambos operam eficientemente em formas convexas e constituem a base de vários motores físicos modernos.

3.1 POLÍGONOS CONVEXOS

Um objeto geométrico é definido como um conjunto não vazio, limitado e fechado de pontos. A propriedade de ser fechado implica que sua fronteira pertence ao próprio conjunto, enquanto a limitação garante que exista uma esfera de raio finito que contenha todos os seus pontos. Um plano, por exemplo, é fechado, mas não limitado.

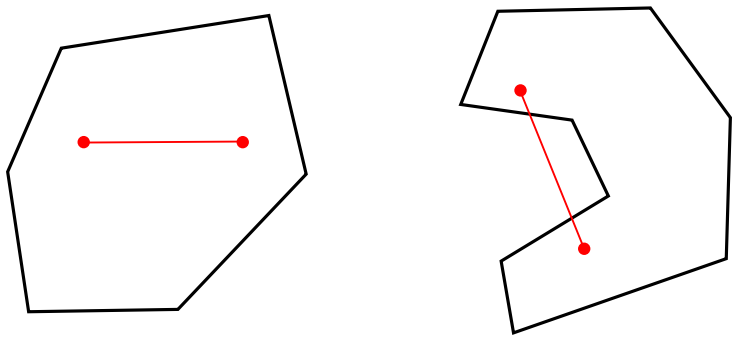

Figura 4 – Comparação entre polígono convexo (à esquerda) e polígono côncavo (à direita).

Uma forma é considerada *convexa* se, para quaisquer dois pontos contidos nessa forma, todo o segmento que os une também estiver contido nela. Uma consequência prática é que, para qualquer linha que atravesse o objeto, esta o intersecta em no máximo dois pontos. Formas não convexas podem ser tratadas como composições de múltiplas partes convexas, o que permite a aplicação direta de algoritmos especializados.

3.2 TEOREMA DO EIXO SEPARADOR (SAT)

O Teorema do Eixo Separador (*Separating Axis Theorem*, SAT) é um dos métodos mais difundidos para detecção de colisão entre polígonos convexos. Além de identificar a presença ou ausência de interseção, o SAT também pode ser utilizado para calcular o *vetor de translação mínima* (*Minimum Translation Vector*, MTV), útil para correções geométricas e resposta física.

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.*

Tal eixo é denominado *eixo separador*. Em termos computacionais, o algoritmo testa um conjunto finito de eixos candidatos. Para polígonos, os candidatos suficientes são as normais das faces (ou arestas, em 2D) de ambos os objetos.

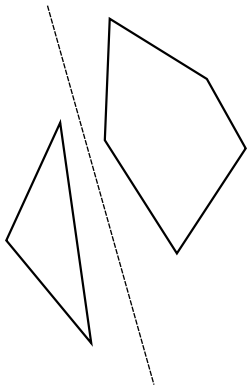

Figura 5 – Eixo separador entre dois polígonos em 2D

Projeção sobre um eixo

Dado um eixo unitário $\hat{n}$ e um conjunto de vértices $\{v\}$ de um polígono, a projeção gera um intervalo escalar $[min, max]$ definido por:

$$min = \min_i(\hat{n} \cdot \vec{v}_i) \tag{3.1}$$

$$max = \max_i(\hat{n} \cdot \vec{v}_i) \tag{3.2}$$

A verificação de sobreposição entre dois intervalos $[min_A, max_A]$ e $[min_B, max_B]$ é dada pela condição:

$$\text{Sobreposição} \iff max_A \geq min_B \quad \wedge \quad max_B \geq min_A \tag{3.3}$$

Se essa condição falhar em qualquer eixo candidato, os objetos estão separados e o algoritmo pode encerrar imediatamente (*early exit*).

Algoritmo de Interseção e MTV

O SAT pode ser estendido para calcular a penetração mínima. Caso todos os eixos apresentem sobreposição, a menor sobreposição encontrada corresponde à magnitude do vetor necessário para resolver a colisão.

Algoritmo 3: SAT com cálculo do MTV

Input: Polígonos convexos A e B
Output: $\vec{d}$ e δ, ou falso
$\vec{d} \leftarrow \vec{0}$
$\delta \leftarrow \infty$
foreach *aresta* $\hat{n}$ *de* A *e* B **do**
  $\hat{n} \leftarrow$ normal unitária de $\hat{n}$
  $p_1 \leftarrow$ Projeção de A
  $p_2 \leftarrow$ Projeção de B
  $\Delta \leftarrow$ sobreposição entre p_1 e p_2
  if $\Delta \leq$ *TOLERÂNCIA* **then**
    return *False*
  if $\Delta < \delta$ **then**
    $\delta \leftarrow \Delta$
    $\vec{d} \leftarrow \hat{n}$
return $\vec{d}$, δ

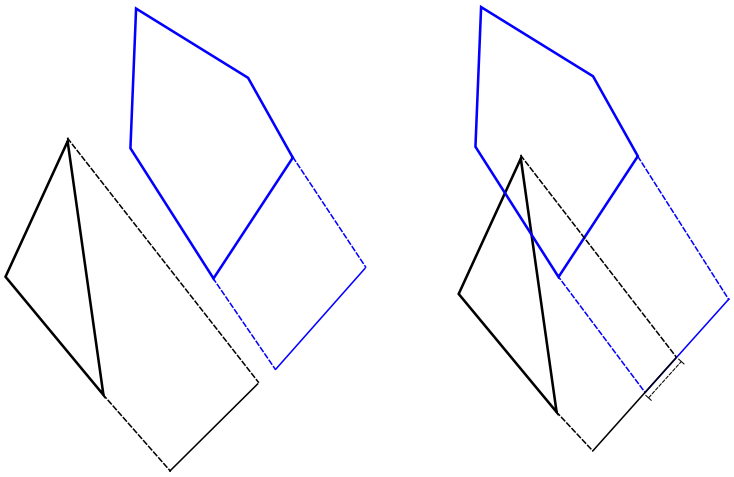

Figura 6 – Projeção e intersecção de eixos

O método possui complexidade linear no número de arestas e é bastante eficiente para polígonos convexos em 2D. Em 3D, entretanto, o número de eixos candidatos cresce significativamente, reduzindo sua praticidade em relação a alternativas como o GJK.

3.3 ALGORITMO GILBERT–JOHNSON–KEERTHI (GJK)

O algoritmo de Gilbert, Johnson e Keerthi (1988) calcula a distância mínima entre dois conjuntos convexos utilizando apenas uma *função de suporte* capaz de retornar o ponto mais distante de um conjunto em uma direção arbitrária. Esse algoritmo é particularmente eficiente em 3D e é amplamente adotado em motores físicos por sua robustez e excelente desempenho.

Soma de Minkowski

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

$$A + B = \{\vec{x} + \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{3.4}$$

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

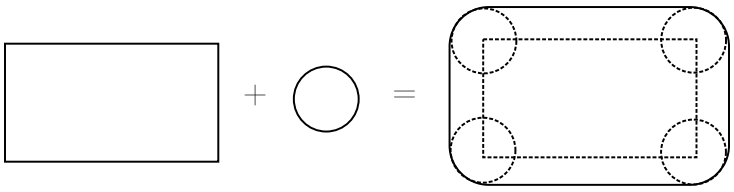

Figura 7 – Soma de Minkowski

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*

$$A - B = \{\vec{x} - \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{3.5}$$

A diferença de Minkowski pode ser pensado como um processo de varredura que calcula o vetor distancia para cada ponto de B em A. 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. Dessa forma, caso a distância entre dois pontos seja zero (interseção), podemos confirmar há 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 $A - B$.*

Executar essa operação exige $|A| * |B| * 2$ subtrações. Isso é significativo porque uma forma é composta de um número infinito de pontos. A grande coisa sobre GJK é que você

realmente não precisa calcular a diferença de Minkowski para todos os vértices. Uma vez que ambas as formas são convexas e definidas por vértices só precisamos realizar esta operação nos vértices mais externos.

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.

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 de 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.

Calcular explicitamente a geometria completa de $A \ominus B$ é computacionalmente proibitivo. O GJK contorna isso utilizando uma *Função de Suporte* $S(\vec{d})$, que retorna o ponto de um corpo mais extremo em uma dada direção $\vec{d}$.

Construção iterativa do simplex

O algoritmo busca determinar se a origem está contida em $A - B$ construindo iterativamente um *simplex* — uma forma geométrica simples (ponto, segmento, triângulo ou tetraedro) — dentro da Diferença de Minkowski.

Algoritmo 4: GJK

Input: Polígonos convexos A e B

Output: Verdadeiro se ocorreu colisão

Escolhe-se uma direção inicial $\vec{d}$;
$p \leftarrow \text{support}(A - B,\ \vec{d})$
if $p \cdot \vec{d} < 0$ **then**
    return *False*
Atualiza-se o simplex com p e calcula-se nova direção $\vec{d}$;
Itera-se até que o simplex contenha a origem ou seja possível concluir ausência de interseção.

O GJK é eficiente e converge rapidamente na prática, sendo mais apropriado que o SAT em cenários tridimensionais e em motores físicos de uso geral.

4 RESPOSTA A COLISÃO

A resposta a colisões é uma etapa fundamental em qualquer sistema de simulação física interativa. Após detectar que dois corpos estão em interpenetração, torna-se necessário aplicar um conjunto de correções que restaurem a plausibilidade física do movimento, evitando instabilidades numéricas.

Este capítulo discute os princípios clássicos e as formulações modernas de resposta a colisão, estabelecendo a relação entre os métodos geométricos de detecção — como o Vetor de Translação Mínima (MTV) — e a abordagem baseada em partículas e restrições proposta por Jakobsen (2001), que fundamenta este trabalho.

4.1 MÉTODOS DINÂMICOS NA SIMULAÇÃO FÍSICA

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

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

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:

Método do Impulso

O *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 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 de Penalidades

Nos *Métodos de Penalidades*, colisões são tratadas como interpenetrações que geram forças de repulsão proporcionais à profundidade de penetração alterando diretamente a aceleração. Essas forças geralmente seguem modelos de mola e amortecimento, 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, essa abordagem é robusta e adequada para sistemas complexos, mas exige a solução de sistemas lineares, tornando sua aplicação onerosa em plataformas Web.

4.2 PROCESSO DE SEPARAÇÃO

A metodologia de Jakobsen (2001) pode ser compreendida como uma precursora da moderna *Position Based Dynamics* (PBD) (MÜLLER et al., 2007). Diferentemente das abordagens que atuam sobre a velocidade ou aceleração, Jakobsen trata colisões como restrições geométricas adicionais que devem ser satisfeitas durante o processo iterativo de relaxação da simulação. Assim, objetos penetrando um ao outro são corrigidos exclusivamente por modificações de posição, de forma estável e sem oscilações numéricas.

A resposta à colisão envolve dois passos principais. O primeiro consiste em separar os elementos geométricos (vértices, arestas ou faces) que se encontram em interpenetração, o que caracteriza um processo estritamente geométrico. O segundo passo corresponde a um processo iterativo de relaxamento, no qual os elementos afetados ajustam suas posições de acordo com as restrições impostas pelo sistema físico.

Para dois objetos convexos A e B em colisão, o esquema de detecção de colisão deve retornar os pontos de contato de cada objeto e o tamanho da penetração. Com essas informações devemos tratar duas configurações possíveis: colisão vértice-vértice ou colisão vértice-aresta.

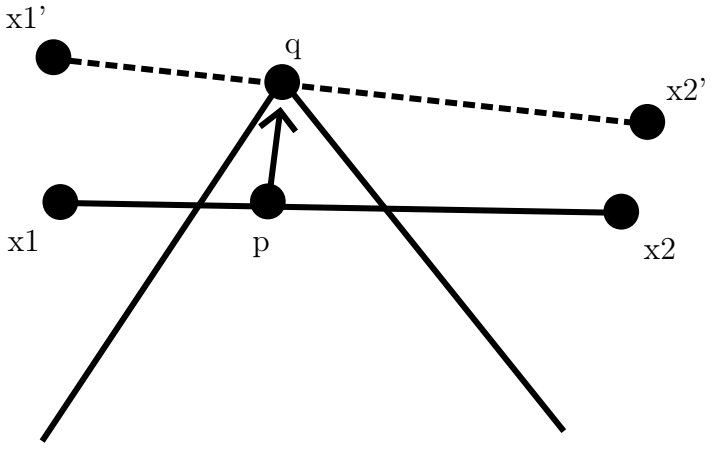

Figura 8 – Processo de separação caso colisão vértice-vértice. Correção das posição para configuração válida. Perceba que como p está mais próximo de x_1, esse vértice é movido mais

Colisão Vértice-Vértice

Se o contato ocorre pontualmente entre duas partículas p e q, a correção é dividida igualmente (assumindo massas iguais):

$$\Delta\vec{p} = -\frac{1}{2}\vec{d}, \qquad \Delta\vec{q} = +\frac{1}{2}\vec{d}. \tag{4.1}$$

Colisão Vértice-Aresta

Esta é a configuração mais comum. Um vértice p de um corpo colide contra uma aresta definida pelas partículas x_1 e x_2 de outro corpo. O ponto de impacto p cai entre dois vértices x_1 e x_2 e o nosso objetivo é corrigir as suas posições para uma configuração válida x_1^*, x_2^*, logo pela equação da reta p pode ser descrito como uma interpolação linear

$$p = \alpha x_1 + (1-\alpha)x_2, \qquad 0 \leq \alpha \leq 1. \tag{4.2}$$

Durante a separação, a correção $\Delta\vec{p}$ deve alterar indiretamente $\vec{x}_1$ e $\vec{x}_2$ de forma proporcional a essa parametrização. A partir da Eq. 4.2, derivamos o valor de α projetando o vetor $\vec{p} - \vec{x}_2$ sobre o vetor da aresta $\vec{x}_1 - \vec{x}_2$. Dessa forma Jakobsen computa as novas posições movendo as partículas proporcionalmente a α:

$$x_1 += \alpha\Delta_p \tag{4.3}$$

$$x_2 += (1-\alpha)\Delta_p \tag{4.4}$$

$$\alpha = \frac{(\vec{p} - \vec{x}_2)\cdot(\vec{x}_1 - \vec{x}_2)}{\|\vec{x}_1 - \vec{x}_2\|^2}. \tag{4.5}$$

Isso garante que a geometria original é preservada e que o ponto $\vec{p}$, definido implicitamente pelos vértices da aresta, é deslocado exatamente pela quantidade desejada.

4.3 ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA)

Enquanto o SAT fornece naturalmente o MTV, o algoritmo GJK apenas informa se há interseção (retornando um *simplex* interno à Diferença de Minkowski). Para obter a profundidade e a normal da colisão necessárias para a resposta física, utiliza-se o **EPA** (*Expanding Polytope Algorithm*).

O algoritmo expande o simplex final do GJK iterativamente até encontrar a fronteira da Diferença de Minkowski mais próxima da origem. A distância entre o ponto mais próximo com a origem é a profundidade de penetração. 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.

Algoritmo 5: EPA

Input: Simplex

Output: $\hat{n}$, δ

while *iterações* $<$ *MAX_ITER* **do**

  e $\leftarrow$ Encontrar aresta mais próxima a origem

  p $\leftarrow$ Calcular novo ponto de suporte na direção da normal de e

  $\delta \leftarrow p \cdot normal(e)$

  if $|\delta - length(e)| < \epsilon$ **then**

    return *normal(e)*, δ

  Adicionar ponto ao simplex

A tolerância ϵ (ex: 10^{-4}) e o limite de iterações são cruciais para evitar loops infinitos em casos de precisão numérica flutuante ou formas curvas aproximadas.

4.4 LIMITAÇÕES

Ao adotar métodos simplificados, abre-se mão de características essenciais de motores físicos completos. Entre as limitações mais relevantes estão:

- **Ausência de conservação precisa de energia e momento**, o que reduz o realismo de certas interações.
- **Incapacidade de simular materiais complexos** (ex.: fricção anisotrópica, torques realistas, elasticidade avançada).
- **Dependência de parâmetros empíricos**, sem interpretação física clara.
- **Menor robustez para geometrias arbitrárias**, especialmente polígonos concavos ou mal escalonados.
- **Dificuldade de lidar com sistemas altamente conectados** (estruturas rígidas, máquinas, esqueletos).

Essas limitações não invalidam o uso das técnicas, mas reforçam a necessidade do uso da aplicação a cenários onde a prioridade é a responsividade, e não a precisão física.

Jittering

Em sistemas baseados em posições a estabilidade depende fortemente do processo de correção de posições. Um dos problemas mais recorrentes é o **jitter**, um tremor ou oscilação indesejada no posicionamento dos corpos, especialmente perceptível quando múltiplas restrições são aplicadas simultaneamente ou quando o sistema é altamente rígido.

Empilhamento

Métodos simplificados têm dificuldade em manter pilhas estáveis de objetos, principalmente quando as correções não são distribuídas de forma global e consistente. O empilhamento tende a "escorregar"ou colapsar devido à falta de amortecimento numérico adequado.

Tunneling

Ocorre quando objetos em alta velocidade atravessam outros sem detectar colisão. Métodos baseados exclusivamente em detecção discreta apresentam maior risco, especialmente quando o passo temporal é grande ou a geometria é fina.

5 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 necessidade de atualizações contínuas e a obrigatoriedade 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, uma simulação contendo 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 quadrática, testar ingenuamente cada par torna-se impraticável 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 estratégias clássicas de otimização aplicadas à detecção e resolução de colisões, com foco na divisão entre *Broad Phase* e *Narrow Phase*, no uso de estruturas espaciais, na adoção de passos temporais fixos e na execução multi-thread de simulações físicas.

5.1 O FECHO CONVEXO

O *Fecho Convexo* (FC) de um conjunto é definido como a menor região convexa que contém todos os seus pontos, sendo frequentemente utilizado como um volume delimitador. Determinar o fecho convexo é um problema recorrente em computação geométrica, especialmente quando se deseja organizar pontos em estruturas mais simples ou acelerar operações posteriores, como testes de colisão.

Em sistemas de simulação física e detecção de colisões, a representação geométrica dos objetos influencia diretamente a eficiência dos cálculos. Formas complexas, com muitos vértices ou superfícies não convexas, tornam tais testes significativamente mais custosos. Uma solução comum é empregar aproximações convexas ou volumes delimitadores que possibilitam testes rápidos sem sacrificar excessivamente a precisão da simulação.

A principal motivação para o uso de volumes delimitadores é que formas mais simples (como caixas ou esferas) permitem testes de sobreposição muito mais baratos do que a geometria original que envolvem. Dessa forma, o FC atua como um primeiro filtro: apenas quando o teste de interseção entre volumes delimitadores retorna positivo é que se procede para verificações mais detalhadas na geometria original. Em muitos casos, o próprio volume delimitador já é suficiente para caracterizar uma colisão.

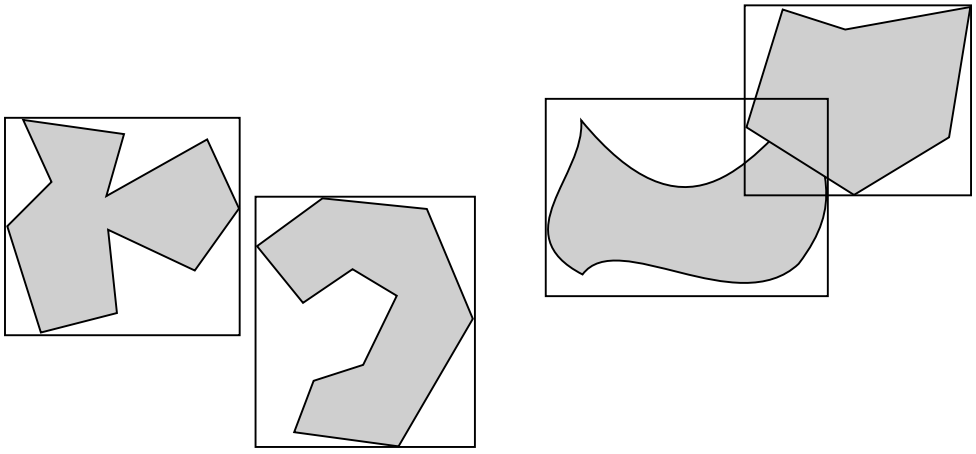

Figura 9 – As figuras à esquerda não tem intersecção de seus volumes delimitadores, logo não podem estar em colisão. Já as figuras da direita estão se sobrepondo, logo **podem** estar em colisão.

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

Caixa Delimitadora Alinhada ao Eixo Coordenado (AABB)

A caixa delimitadora alinhada aos eixos (Axis-Aligned Bounding Box, AABB) é um dos volumes delimitadores mais utilizados. Trata-se de um paralelepípedo (ou retângulo, em 2D) cujas faces são paralelas aos eixos do sistema de coordenadas. Seu teste de interseção é extremamente simples:

Algoritmo 6: Teste de Interseção AABB

Input: A e B: volumes AABB

Output: Verdadeiro se houver colisão

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

if $A.y_{max} < B.y_{min}$ ***ou*** $A.y_{min} > B.y_{max}$ **then**
  return *False*

if $A.z_{max} < B.z_{min}$ ***ou*** $A.z_{min} > B.z_{max}$ **then**
  return *False*

return *True*

AABBs são eficientes, porém perdem precisão quando o objeto sofre rotações, pois a caixa permanece alinhada aos eixos globais.

Caixa Orientada (OBB)

Uma Caixa Orientada (Oriented Bounding Box, OBB) é uma caixa retangular que pode estar arbitrariamente rotacionada em relação aos eixos do sistema de coordenadas. É definida por um ponto central c, por três vetores ortogonais $\hat{u}_i$ que compõem sua orientação e por semi-extensões e_i:

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

OBBs geralmente oferecem melhor ajuste, especialmente para objetos alongados ou rotacionados, reduzindo falsos positivos. Porém, o teste de interseção é mais caro que o das AABBs, o que torna seu uso preferível em cenas com número reduzido de objetos ou em simulações nas quais a precisão de ajuste é particularmente importante.

Esferas e Elipsoides

As **esferas** constituem o volume delimitador mais simples, definidas apenas por um centro c e um raio r:

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

Testes de colisão entre esferas são extremamente rápidos, porém inadequados para objetos de proporções irregulares. Elipsoides oferecem melhor ajuste, mas aumentam o custo de teste. Por isso, essas formas são frequentemente utilizadas em fases preliminares da detecção, ou como nós intermediários em hierarquias de volumes delimitadores (BVH).

Quickhull

O *Quickhull* é um algoritmo para o cálculo do fecho convexo de um conjunto finito de pontos em qualquer dimensão, adotando uma estratégia de divisão e conquista semelhante

ao *quicksort* (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 7: Quickhull 2D

Input: Polígono Convexo

Output: Lista dos vértices do fecho convexo

1 Encontre os pontos de menor e maior coordenada em x; eles pertencem ao fecho 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 Para cada subconjunto, encontre o ponto mais distante da linha; ele forma um triângulo que exclui pontos interiores.
4 Repita recursivamente os dois passos anteriores nas duas linhas formadas pelos dois novos lados do triângulo.
5 O processo termina quando todos os subconjuntos estão vazios.

O Quickhull apresenta complexidade média $O(n \log n)$ em 2D, podendo chegar a $O(n^2)$ em casos degenerados. Em 3D, adapta-se a construções poliedrais mais complexas, mantendo o mesmo princípio recursivo.

5.2 BROAD PHASE

A fase de *Broad Phase* tem como objetivo descartar rapidamente pares de objetos que seguramente não estão colidindo. Segundo Ericson (2004), "nada é mais rápido do que não ter que realizar uma tarefa". Portanto, a melhor otimização é reduzir a quantidade de trabalho o mais cedo possível.

Como objetos só podem colidir com outros que estejam fisicamente próximos, a Broad Phase utiliza consultas espaciais para identificar apenas aqueles que compartilham regiões semelhantes do espaço. O teste mais simples usado nesta etapa é o teste de interseção booleana entre volumes delimitadores primitivos, devido ao seu baixo custo computacional.

Algoritmo 8: Broad Phase generalizada

```
Output: Pares de objetos potenciais para colisão
vistos ← {}
pares_contato ← {}
foreach bodyA em bodies do
    candidates ← consultar objetos próximos de bodyA
    foreach bodyB em candidates do
        if par {bodyA, bodyB} já foi visto then
            continue
        marcar par como visto
        if teste de interseção barata then
            pares_contato ← {bodyA, bodyB}
return pares_contato
```

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. Células muito pequenas geram atualizações excessivas.
2. Células grandes demais fazem com que muitos objetos sejam agrupados, reduzindo a eficácia da Broad Phase.
3. Objetos muito complexos demandam subdivisão para melhorar a qualidade dos testes.
4. Cenários mistos exigem grades hierárquicas ou abordagens híbridas.

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

5.3 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.

5.4 FÍSICA COM PASSO DE TEMPO FIXO

A forma mais ingênua de simular física é utilizar o tempo decorrido entre quadros (*delta time*) como passo de simulação. Embora simples, isso introduz instabilidade numérica: simulações podem divergir em altas taxas de quadros, apresentar *tunneling* e se comportar de maneira não determinística. Essa abordagem trás um passo variável que será executada mais rápido ou mais lento dependendo do computador do usuário.

Algoritmo 9: 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()
```

Para garantir estabilidade, utiliza-se um passo fixo de simulação e um acumulador de tempo:

Algoritmo 10: 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()
```

5.5 SIMULAÇÃO FÍSICA MULTI-THREAD

Em programas interativos tradicionais a lógica, a física e a renderização são executadas em uma única **thread**. Nesse modelo, a renderização só ocorre após a conclusão da etapa de física, e qualquer uma das etapas pode se tornar gargalo.

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.

Para solucionar esse problema, a simulação física pode ser movida para um *thread* dedicado. A thread principal renderiza continuamente utilizando o estado físico mais recente, enquanto o thread secundário calcula atualizações físicas em paralelo.

INSERIR NOVO DIAGRAMA

Essa separação permite:

- melhor utilização de múltiplos núcleos;
- redução da latência na renderização;
- maior taxa de quadros mesmo em cenas fisicamente complexas;
- desacoplamento total entre física e renderização.

Essa arquitetura é essencial em jogos modernos e simulações interativas, especialmente em ambientes Web utilizando Web Workers.

6 EXPERIMENTOS

Este capítulo apresenta os experimentos realizados com o objetivo de avaliar o desempenho, a estabilidade e a precisão do sistema desenvolvido. 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.

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

6.1 CONFIGURAÇÃO DOS CENÁRIOS

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

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.

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.

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.

6.2 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.

6.3 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

7 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.

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.

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.

MÜLLER, M. et al. Position based dynamics. **Journal of Visual Communication and Image Representation**, v. 18, n. 2, p. 109–118, 2007. ISSN 1047-3203. Disponível em: https://www.sciencedirect.com/science/article/pii/S1047320307000065.