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

"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 une 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 o Havok (**??**) popularizou o uso profissional de física em jogos de grande porte, oferecendo um sistema robusto de colisões e restrições. O NVIDIA PhysX (NVIDIA Omniverse,) 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 (JAKOBSEN, 2001) 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 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:
> $$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt} = m\frac{d^2\vec{x}}{dt^2},$$
> onde $\vec{x}$ é a posição do corpo, $\vec{v}$ é sua velocidade e t é o tempo.
>
> **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:

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

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

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

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 quatérnio;
- **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.

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, tipicamente baseados em partículas, são preferidos em relação a métodos mais complexos, como por exemplo os Métodos de Elementos Finitos (FEM).

Sistemas Massa-Mola

A abordagem mais comum para simular deformações em jogos e aplicações interativas é o modelo Massa-Mola. 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:

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

onde k_s é a constante de rigidez, 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:

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

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 grade. 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: Conectam partículas vizinhas diretas (horizontal e verticalmente). Elas resistem à tração e compressão básicas, mantendo a integridade da malha.

2. Molas de Cisalhamento: 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: 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:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t,$$
$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_t \Delta t.$$

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:

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t,$$
$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \Delta t.$$

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

Integração de Verlet

Verlet 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:

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

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

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

Esse é o método usado por Jakobsen (2001), por ser um método 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) descreve uma restrição geométrica simples para compor estruturas complexas, a restrição linear (ou restrição de distância) mantém uma distância fixa d entre duas partículas i e j. Dessa forma o conjunto de partículas deve satisfazer a todo instante uma coleção de equações na forma:

$$|\vec{x_i} - \vec{x_j}| = d. \tag{2.1}$$

Mesmo que as posições das partículas estejam inicialmente corretas, depois de um passo de simulação a distância entre elas pode se tornar inválida (devido a forças externas, por exemplo). Para corrigir sua distância devemos movê-las (projetar) de forma a satisfazer Eq. 2.1, como ilustrado na Figura 1.

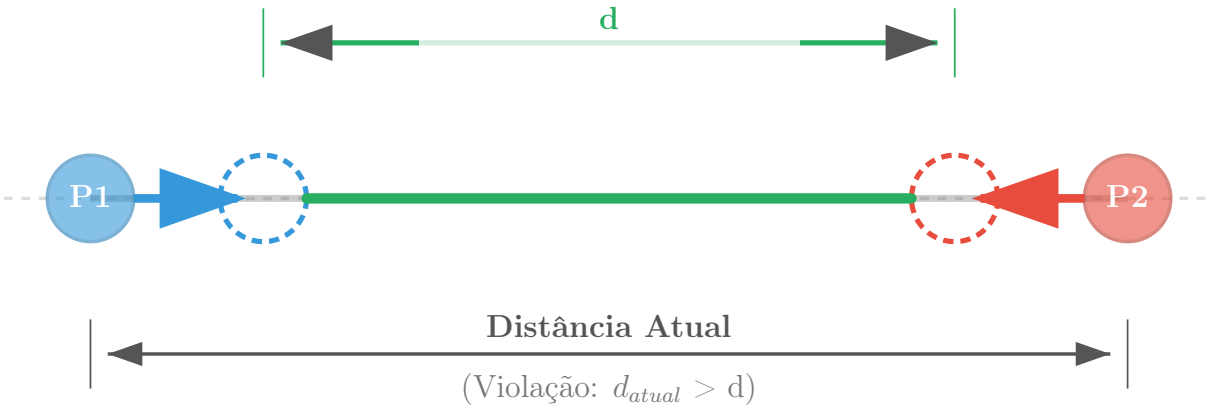


Figura 1 – A distância atual entre as partículas P1 e P2 estão inválidas e devem ser corrigidas por projeção.

Jakobsen (2001) também descreve um esquema de articulação (dobradiça) fazendo que objetos compartilhem a mesma partícula como ilustrado na Figura 2. Dessa forma, seria possível modelar um corpo humanoide, Jakobsen (2001) também mostra que para realismo adicional pode-se limitar o ângulo de rotação de articulações através de restrições angulares satisfazendo o produto interno abaixo:

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

Um método alternativo e simples que Jakobsen (2001) também mostra é usar restrições de distância auxiliares, que não foi usada neste trabalho, onde a distância entre as extremidades i e j não seja maior que um valor d_{min}:

$$|\vec{x}_i - \vec{x}_j| > d_{min}$$

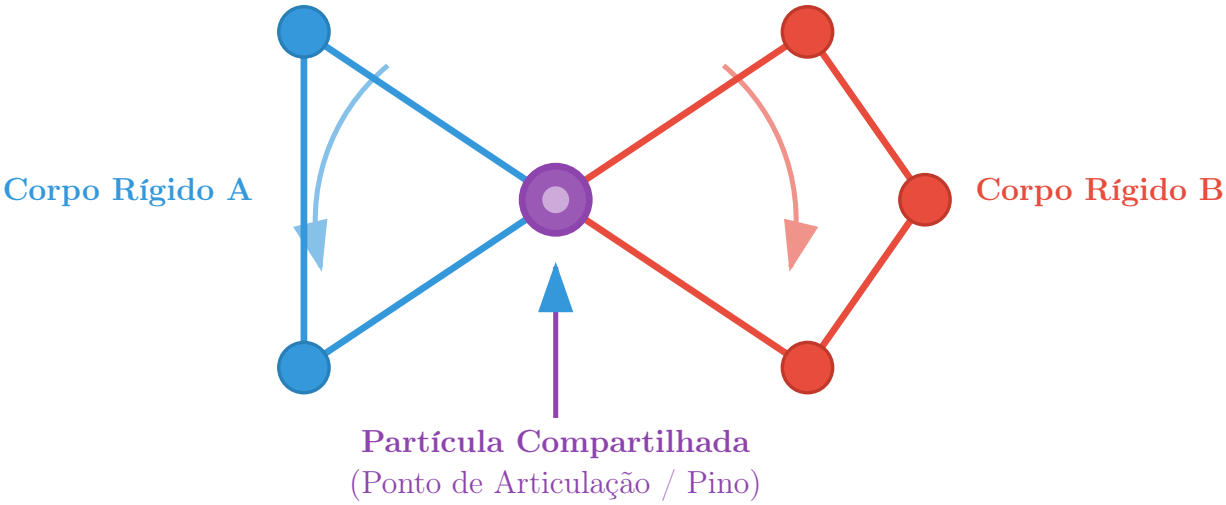


Figura 2 – A articulação é formada simplesmente fazendo os dois corpos compartilharem a mesma partícula

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 elas 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 relaxamento de *Gauss-Seidel*, 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 satisfaz todas restrições. Quanto maior o número de iterações, mais rápido o sistema irá convergir para solução ideal e também a animação irá parecer mais rígida para o usuário.

Deve-se encontrar o movimento mínimo que satisfaça cada restrição, para restrição linear podemos fazer como no Algoritmo 1 que irá separar ou aproximar as partículas de

tal forma que satisfaçam a distância d. Onde ϵ representa o tamanho de passo local para

Algoritmo 1: Relaxamento para restrição linear

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

convergência para solução ideal. Quando $\epsilon = 1$ as partículas são movidas imediatamente para posição correta. 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.

Jakobsen (2001) 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.

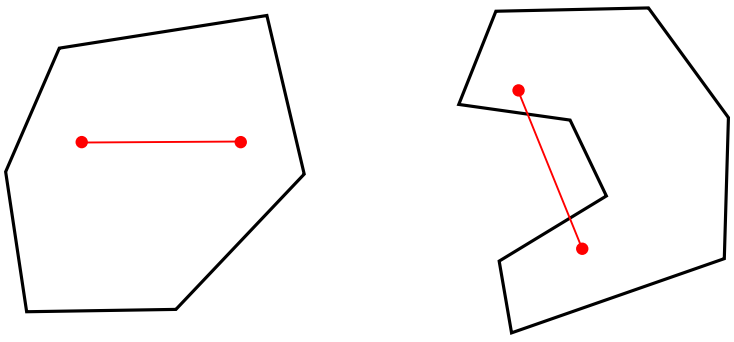


Figura 3 – Comparação entre polígono convexo (à esquerda) e polígono côncavo (à direita).

Uma forma é considerada convexa se, para quaisquer dois pontos contidos nessa forma, todo o segmento que os une também estiver contido nela, como ilustrado na Figura 3. 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 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 (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. Ao traçar raios paralelos sob dois objetos (a partir de uma fonte luminosa, por exemplo) se as sombras formadas estiverem separadas então não há colisão, repita esse processo ao redor das formas, caso não encontre uma sombra que esteja separada uma da outra os objetos estão em colisão. O SAT 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, como ilustrado na Figura 4. 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.

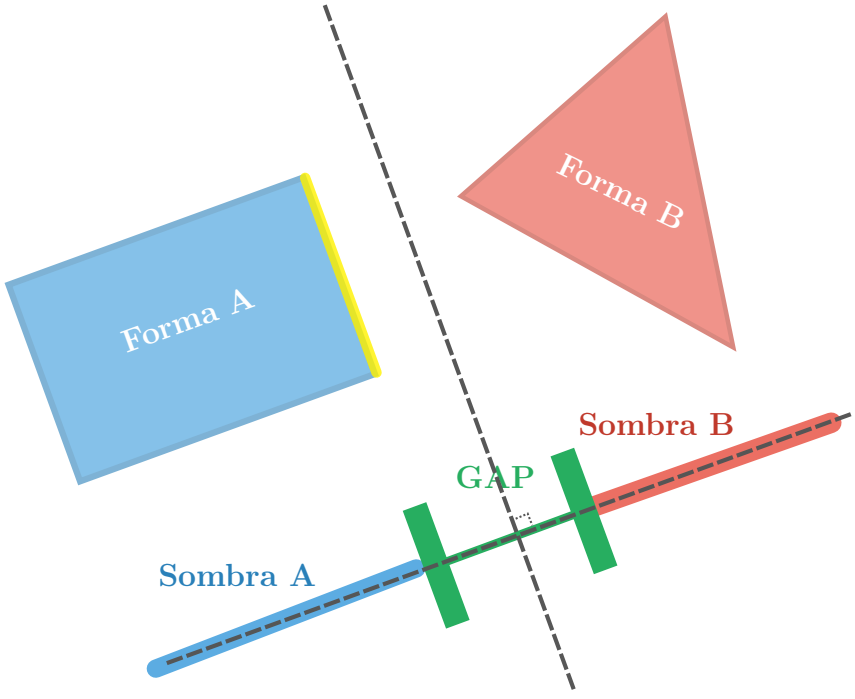


Figura 4 – Face escolhida em amarelo, eixo separador perpendicular a face.

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:

$$min = \min_i(\hat{n} \cdot \vec{v}_i)$$
$$max = \max_i(\hat{n} \cdot \vec{v}_i).$$

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

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

como ilustrado na Figura 4. Se essa condição falhar em qualquer eixo candidato, os objetos estão separados e o algoritmo pode encerrar imediatamente.

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, como mostrado no Algoritmo 2 , onde ϵ é um

Algoritmo 2: SAT com cálculo do MTV

```
Input: Polígonos convexos A e B
Output: d⃗ e δ, ou falso
d⃗ ← 0⃗
δ ← ∞
foreach aresta n̂ de A e B do
    n̂ ← normal unitária de n̂
    p1 ← Projeção de A
    p2 ← Projeção de B
    Δ ← sobreposição entre p1 e p2
    if Δ ≤ ε then
        return False
    if Δ < δ then
        δ ← Δ
        d⃗ ← n̂
return d⃗, δ
```

valor numérico para tratar erros de operação flutuante e deve-se ser um valor baixo. 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 original de Gilbert, Johnson e Keerthi (1988) calcula a distância mínima entre dois conjuntos convexos, neste trabalho faremos uma versão simplificada, apresentada por Muratori (2016), que realiza um teste Booleano GJK (BGJK) para o caso de dois objetos estarem em intersecção, com otimização usando regiões de Voronoi. Esse

algoritmo funciona para ambientes 2D, é particularmente eficiente em 3D e é amplamente adotado em motores físicos por sua robustez e excelente desempenho.

O algoritmo GJK fundamenta-se na soma de Minkowski entre dois conjuntos convexos A e B, definida como

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

onde $\vec{x}$ e $\vec{y}$ representam vetores posição associados a pontos pertencentes aos conjuntos A e B, respectivamente.

Do ponto de vista geométrico, a soma de Minkowski pode ser interpretada como o conjunto de todos os pontos obtidos ao transladar o objeto B por cada vetor posição pertencente a A, mantendo sua forma e orientação, ou seja, faz-se uma cópia do objeto B centrado em cada ponto de A, como ilustrado na Figura 5.

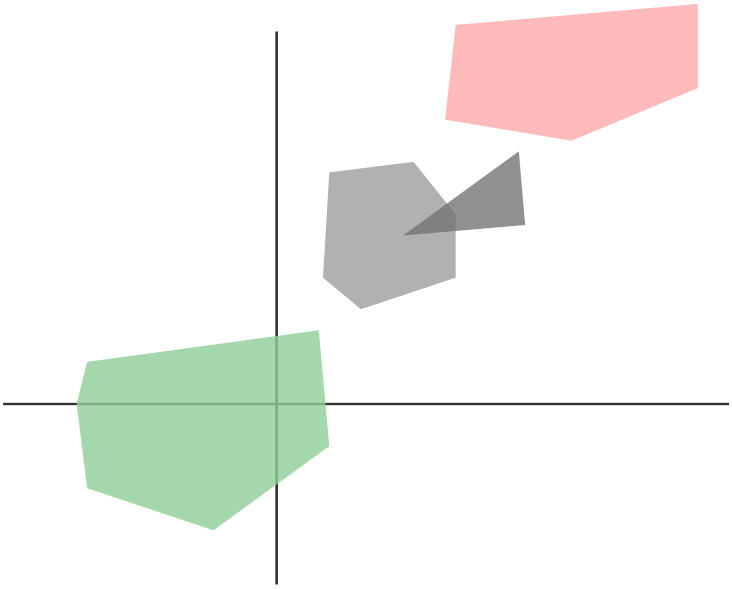


Figura 5 – Em vermelho a soma de Minkowski, em verde a diferença de Minkowski

Uma propriedade muito útil da soma de Minkowski é o fato de que a soma de dois objetos convexos é um objeto convexo. Para um objeto A, usamos a notação $-A$ para denotar o reflexo de A sobre a origem O. A diferença de Minkowski $A - B$ pode ser obtida calculando a soma de Minkowski de A e $-B$

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

que pode ser pensado como um processo de varredura que calcula o vetor distância para cada ponto de B em A. Neste trabalho, usaremos esse termo para referir a essa operação quando necessário.

A diferença de Minkowski também é chamada de configuração do espaço de obstáculos (CSO). Esse espaço possui uma propriedade muito útil de que A e B se intersectam se, e somente se, sua CSO contém a origem. Isso se deve ao fato de que se existe um ponto

em comum entre A e B onde a diferença de Minkowski é igual a zero, logo, esse ponto coincide com a origem O.

Assim o objetivo do BGJK reduz para determinar se a CSO contém ou não a origem. A grande coisa do BGJK é que você não precisa calcular todo CSO, apenas uma amostra do CSO modificando iterativamente um Simplex dentro da OSC até que o Simplex contenha a origem ou pare por não ser possível a origem estar contida no Simplex.

Um Simplex é definido como a forma mais simples possível em um espaço de k-dimensão. Dessa forma um ponto é 0-simplex, um segmento de reta é 1-simplex, um triângulo é 2-simplex, um tetraedro é 3-simplex, como na Figura 6.

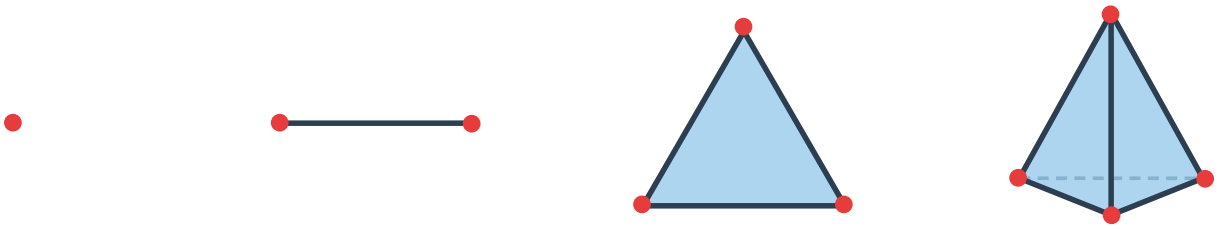


Figura 6 – Da esquerda para direita 0-simplex, 1-simplex, 2-simplex e 3-simplex

A construção de um Simplex dentro da região CSO usa uma função de suporte que é definida tal que: seja A um conjunto qualquer, o suporte de A na direção $\vec{d}$ retorna o ponto que pertence a superfície de A mais distante da origem na direção $\vec{d}$, esse ponto é chamado de ponto de suporte. Matematicamente fazemos $S_A(\vec{d}) = max\{v \cdot a : a \in A\}$.

Também é possível realizar aritmética com função suporte

$$S_{A+B}(\vec{d}) = S_A(\vec{d}) + S_B(\vec{d})$$
$$S_{-A}(\vec{d}) = -S_A(-\vec{d}),$$

dessa forma, seja D a diferença de Minkowski de dois objetos convexos A e B, o suporte de D na direção $\vec{d}$ é a diferença dos suportes de A e B: $S_{A-B}(\vec{d}) = S_A(\vec{d}) - S_B(-\vec{d})$, como ilustrado pela Figura 7. Veremos no final deste capítulo, mesmo o CSO possuindo um infinito número de pontos o GJK consegue trabalhar em seu espaço calculando pontos sobre demanda.

Linahan (2015) mostra o BGJK (MURATORI, 2016) como no Algoritmo 3, inicializando um simplex Q com um ponto de suporte S numa direção aleatória do CSO (é comum ser escolhido a direção do centro de A para B). Então S é usado para gerar uma nova direção $\vec{d}$ oposta a S, isso é feito para maximizar a área do Simplex e as chances de conter a origem. O loop principal consiste em três etapas principais: primeiro calcular um novo ponto de suporte na direção atual e verificar se esse ponto ultrapassou a origem (S $\cdot \vec{d} < 0$), se sim a origem não pode estar contida no Simplex Q, caso contrário adicionamos S ao Simplex e continuamos tentando incluir a origem.

Por fim, chamamos a rotina *DoSimplex* que é responsável por: calcular se a origem está contida ou não no Simplex Q usando regiões de Voronoi, remover pontos antigos do

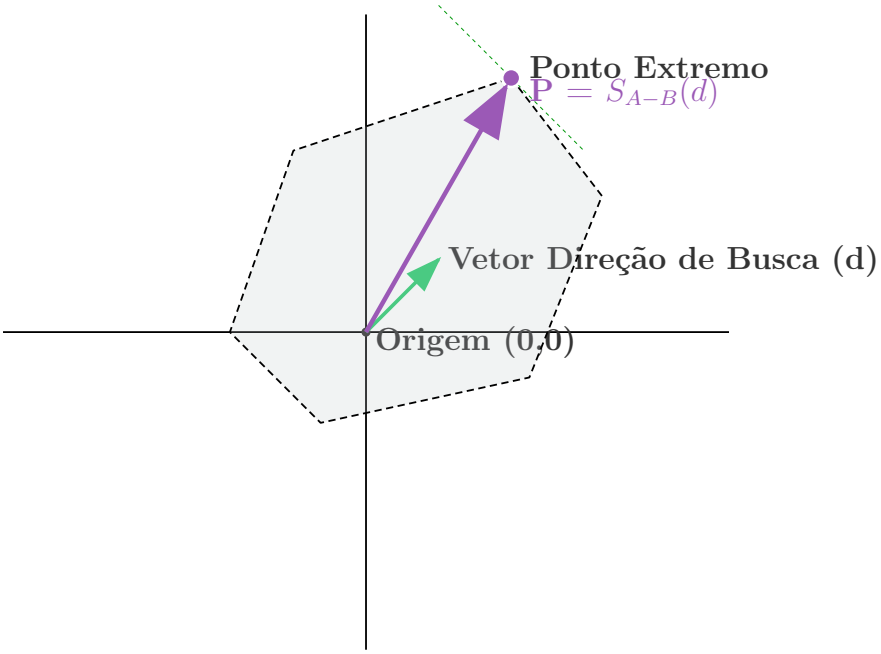


Figura 7 – A função suporte na forma implícita $A - B$ ao longo da direção $\vec{d}$.

Algoritmo 3: Boolean GJK

Input: Polígonos convexos A e B
Output: True se colidindo, False caso contrário
S $\leftarrow S_{CSO}$(Direção aleatória)
Q $\leftarrow$ {S}
$\vec{d} \leftarrow -S$
while *iterações* $<$ *MAX_ITER* **do**
  S $\leftarrow S_{CSO}(\vec{d})$
  if $S \cdot \vec{d} < 0$ **then**
    return False
  $Q \leftarrow Q \cup \{S\}$
  if *DoSimplex(Q,* $\vec{d}$*)* **then**
    return True
return False

simplex e procurar por uma nova direção que maximize as chances do próximo ponto de suporte conter a origem. Para cada dimensão é feito uma conta diferente, neste trabalho iremos trabalhar apenas com objetos em 2D, por isso trataremos apenas dos casos 1-simplex e 2-simplex, Linahan (2015) realiza a prova a corretude e também mostra para 3-simplex. Para detalhar essa rotina usaremos uma conveção de nomear os pontos do Simplex em ordem alfabética, com o ponto mais recente sendo A.

1-Simplex

No caso de 2 pontos, Q={A,B} é um 1-simplex com 3 regiões de Voronoi: dois semi-espaços e uma faixa. Se o CSO contiver a origem quando n = 1, a região de Voronoi da faixa deve conter a origem.

2-Simplex

No caso de 3 pontos, Q = {A, B, C} é um 2-simplex com 8 regiões de Voronoi: uma para cada um dos três vértices, uma para cada uma das três arestas e duas para as regiões acima e abaixo do triângulo. Se o CSO contém a origem quando n = 2, então as regiões de Voronoi AC, AB, ABC ou ACB devem conter a origem.

4 RESPOSTA A COLISÕES

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 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 (2001), é importante contextualizar outras categorias amplamente utilizadas em motores físicos modernos:

Método do Impulso

O Método do Impulso trata 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 Baseado em Restrições 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 (2001) trata colisões como restrições geométricas adicionais que devem ser satisfeitas durante o processo iterativo de relaxamento da simulação. Assim, objetos penetrando um ao outro são corrigidos exclusivamente por modificações de posição.

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, o tamanho da penetração e a direção $\vec{d}$ de separação. Com essas informações devemos tratar duas configurações possíveis: colisão vértice-vértice ou colisão vértice-aresta.

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 $\vec{x}_1$ e $\vec{x}_2$ de outro corpo. O ponto de impacto p cai entre dois vértices $\vec{x}_1$ e $\vec{x}_2$ e o nosso objetivo é corrigir as suas posições para uma configuração válida $\vec{x}'_1$, $\vec{x}'_2$ de tal forma que garanta uma coerência geométrica, como mostra na Figura 8. Logo, pela equação da reta, p pode ser descrito como uma interpolação linear

$$p = \alpha\vec{x}_1 + (1-\alpha)\vec{x}_2, \qquad 0 \leq \alpha \leq 1, \tag{4.2}$$

ou seja, quando $\alpha = 0, p = x_2$, quando $\alpha = 1, p = x_1$.

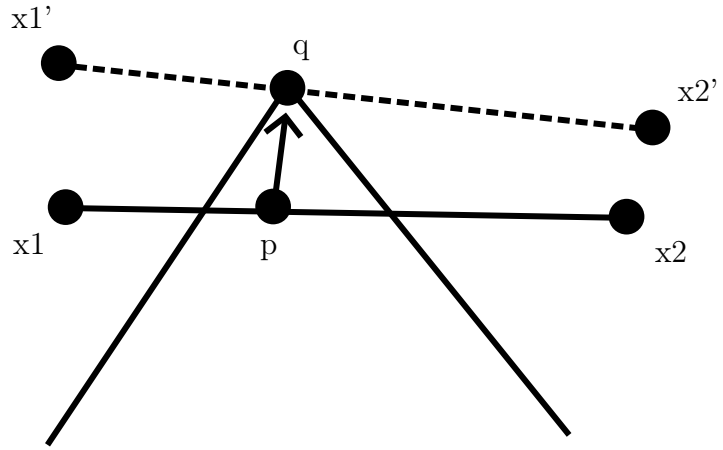


Figura 8 – Processo de separação caso colisão vértice-aresta. Correção das posição para configuração válida. Perceba como p está mais próximo de x_1, esse vértice é movido mais

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 calculando quanto o ponto de contato p está ao "longo" da aresta que vai do ponto $\vec{x}_2 \to \vec{x}_1$. Dessa forma, Jakobsen (2001) computa as novas posições movendo as partículas proporcionalmente a α:

$$\begin{aligned} \vec{x}_1 &+= \alpha\Delta_p \\ \vec{x}_2 &+= (1-\alpha)\Delta_p \\ \alpha &= \frac{(\vec{p}-\vec{x}_2)\cdot(\vec{x}_1-\vec{x}_2)}{\|\vec{x}_1-\vec{x}_2\|^2}. \end{aligned}$$

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 BGJK 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 Algoritmo de Expansão de Politopos (EPA).

O Algoritmo 4 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 a normal da face mais próxima da origem, porém um *simplex* não precisa conter nenhuma das faces do polígono original, o que pode levar ao cálculo de uma normal incorreta. A

Algoritmo 4: EPA

```
Input: Simplex
Output: n̂, δ
while iterações < MAX_ITER do
    e ← Encontrar aresta mais próxima a origem
    p ← Calcular novo ponto de suporte na direção da normal de e
    δ ← p · normal(e)
    if |δ − length(e)| < ϵ then
        return normal(e), δ
    Adicionar ponto ao simplex
```

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 côncavos 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 precisão numérica 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: *Broad Phase* e *Narrow 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 OBJETOS ENVOLVENTES

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 que possibilitam testes rápidos sem sacrificar excessivamente a precisão da simulação.

A principal motivação para o uso de objetos envolventes é que formas mais simples (como caixas ou esferas) permitem testes de sobreposição muito mais baratos do que a geometria original que envolvem, como ilustrado na Figura 9. Dessa forma, objetos envolventes atuam como um primeiro filtro: apenas quando o teste de interseção entre objetos envolventes 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.

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

- Testes de interseção de baixo custo

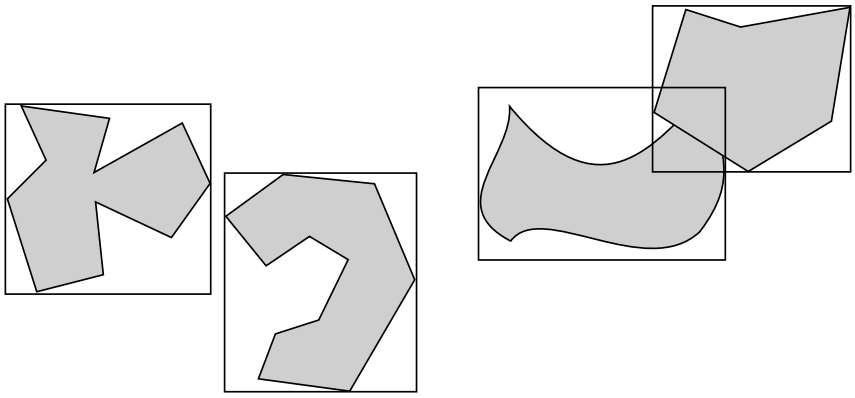
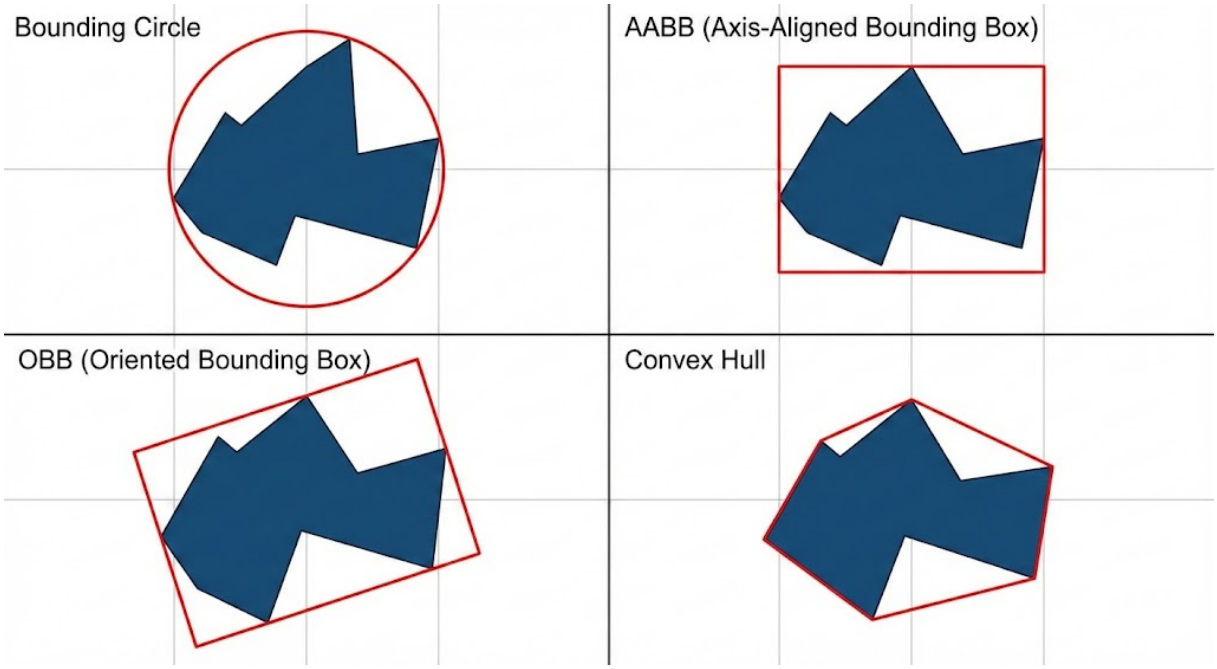


Figura 9 – As figuras à esquerda não tem intersecção de seus volumes delimitadores, logo não podem estar em colisão. Já as figuras da direita estão se sobrepondo, logo podem estar em colisão.

- Ajuste preciso
- Cálculo econômico
- Fácil de girar e transformar
- Consome pouca memória

como ilustrado na Figura 10.

Figura 10 – Comparação entre formas de fecho convexo



Esfera Delimitadora

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

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

Testes de colisão entre esferas são 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).

Caixa Delimitadora Alinhada ao Eixo Coordenado (AABB)

A caixa delimitadora alinhada aos eixos (AABB) foi usado por ser um dos volumes delimitadores mais comuns. Trata-se de um paralelepípedo (ou retângulo, em 2D) cujas faces são paralelas aos eixos do sistema de coordenadas. Seu teste de interseção é simples como no Algoritmo 5. Neste trabalho foram usadasa AABBs por serem simples de imple-

Algoritmo 5: Teste de Intersecção 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*

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

Caixa Delimitadora Orientada (OBB)

Uma Caixa Delimitadora Orientada (OBB) é uma caixa retangular que pode estar arbitrariamente rotacionada em relação aos eixos do sistema de coordenadas. A representação mais comum é feita por um ponto central c, por três vetores ortogonais $\hat{u}_i$ que compõem sua orientação e por extensões e_i, que são assumidas serem positivas.

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 e geralmente é feito usando o SAT.

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.

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 como no Algoritmo 5. Apresenta complexidade média $O(n \log n)$ em 2D, podendo chegar a

Algoritmo 6: 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.

$O(n^2)$ 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 próximas do espaço. O teste mais simples usado nesta etapa é o teste de interseção booleana entre volumes delimitadores primitivos como no Algoritmo 7, devido ao seu baixo custo computacional.

Algoritmo 7: Broad Phase generalizada

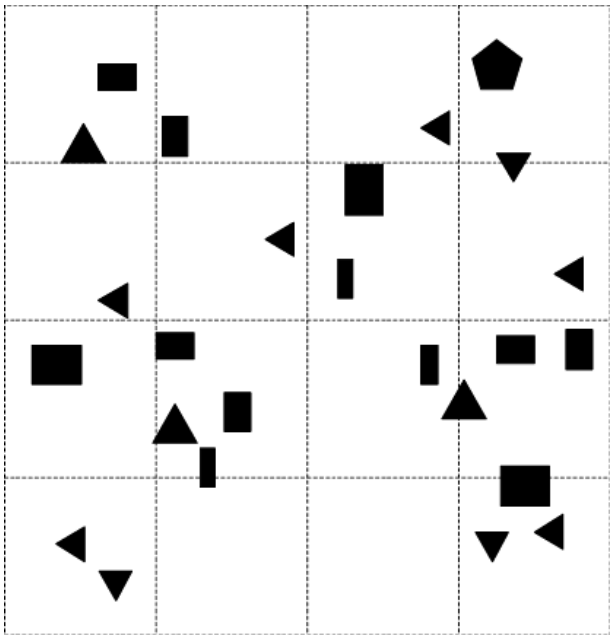
```
Output: Pares de objetos potenciais para colisão
vistos ← {}
pares_contato ← {}
foreach bodyA em bodies do
    candidates ← consultar objetos próximos de bodyA
    foreach bodyB em candidates do
        if par {bodyA, bodyB} já foi visto then
            continue
        marcar par como visto
        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 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, como ilustrado na Figura 11.

Figura 11 – Divisão espacial em grade uniforme



Devido à uniformidade da grade, acessar uma célula correspondente a uma determi-

nada 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 como passo de simulação, como ilustrado no Algoritmo 8. 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.

Para garantir estabilidade, utiliza-se um passo fixo de simulação e um acumulador de tempo como no Algoritmo 9.

Algoritmo 8: 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()
```

Algoritmo 9: 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.

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, como ilustrado na Figura 12. 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* dedicada. A *thread* principal renderiza continuamente utilizando o estado físico mais recente, enquanto o *thread* secundário calcula atualizações físicas em paralelo, como ilustrado na Figura 12.

Essa separação permite:

- melhor utilização de múltiplos núcleos;

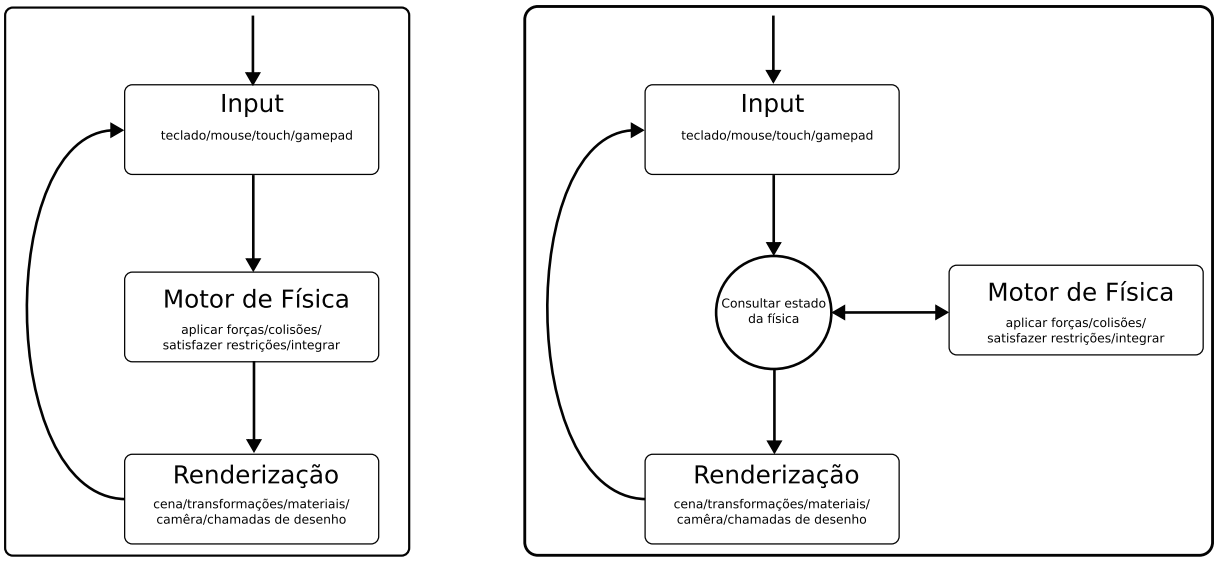


Figura 12 – À esquerda esquema tradicional single-thread. À direita esquema com simulação rodando numa thread dedicada a passo fixo.

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

Dessa forma, a 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 SAT e 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.

- 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.

A execucação da física em uma thread separada apresentou estabilidade na renderização.

6.2 RESULTADOS E DISCUSSÃO

Os resultados obtidos indicam que o método de Jakobsen (2001) 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 Verlet, 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.

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.

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.

LINAHAN, J. **A Geometric Interpretation of the Boolean Gilbert-Johnson-Keerthi Algorithm**. 2015. Disponível em: https://arxiv.org/abs/1505.07873.

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.

MURATORI, C. **Implementing GJK (2006)**. 2016. https://caseymuratori.com/blog_0003. Acessado em: 12 de maio de 2016.

NVIDIA Omniverse. **PhysX SDK**. https://github.com/NVIDIA-Omniverse/PhysX. BSD-3 license, Accessed: 2025-11-25.