

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.

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

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

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

SUMÁRIO

1	INTRODUÇÃO	8
1.1	CONTEXTUALIZAÇÃO DO PROBLEMA	8
1.2	MOTORES FÍSICOS E EVOLUÇÃO HISTÓRICA	9
1.3	APLICAÇÕES WEB E MOTIVAÇÃO TECNOLÓGICA	9
1.4	JUSTIFICATIVA	10
1.5	OBJETIVOS	10
1.6	ESTRUTURA DO TRABALHO	10
2	ANIMAÇÃO BASEADA EM FÍSICA	12
2.1	CONCEITOS E DEFINIÇÕES	12
2.2	REPRESENTAÇÃO DE CORPOS RÍGIDOS	13
2.3	DINÂMICA DE PARTÍCULAS	14
3	MÉTODO SIMPLIFICADO DE JAKOBSEN	15
3.1	MÉTODOS NUMÉRICOS EM SIMULAÇÃO	15
3.1.1	Método de Euler	15
3.1.2	Método Semi-implícito de Euler	15
3.1.3	Integração de Verlet	16
3.2	RESTRIÇÕES GEOMÉTRICAS	16
3.2.1	Restrição Linear	16
3.2.2	Restrição de Revolução	17
3.2.3	Restrição Angular	17
3.3	RESOLVENDO RESTRIÇÕES CONCORRENTES POR RELAXAMENTO	18
4	DETECÇÃO DE COLISÕES	19
4.1	POLÍGONOS CONVEXOS	19
4.2	TEOREMA DO EIXO SEPARADOR (SAT)	19
4.2.1	Projeção sobre um eixo	20
4.2.2	Teste de interseção booleana	20
4.2.3	Vetor de Translação Mínima (MTV)	21
4.3	ALGORITMO GILBERT–JOHNSON–KEERTHI (GJK)	21
4.3.1	Soma de Minkowski	21
4.3.2	Função de Suporte	22
4.3.3	Construção iterativa do simplex	23

5	RESPOSTA A COLISÃO	24
5.1	FUNDAMENTOS DA RESPOSTA A COLISÃO	24
5.2	MÉTODOS DINÂMICOS NA SIMULAÇÃO FÍSICA	24
5.2.1	Método do Impulso	24
5.2.2	Método de Penalidades	25
5.2.3	Método Constraint-Based com Multiplicadores de Lagrange	25
5.3	PROCESSO DE SEPARAÇÃO	25
5.4	ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA)	26
5.5	DESAFIOS	27
5.5.1	Jittering	27
5.5.2	Empilhamento	27
5.5.3	Tunneling	28
5.6	LIMITAÇÕES	28
6	OTIMIZAÇÕES	29
6.1	O FECHO CONVEXO	29
6.1.1	Caixa Delimitadora Alinhada ao Eixo Coordenado (AABB)	30
6.1.2	Caixa Orientada (OBB)	31
6.1.3	Esferas e Elipsoides	31
6.1.4	Quickhull	31
6.2	BROAD PHASE	32
6.2.1	Grade uniforme	33
6.3	NARROW PHASE	33
6.4	FÍSICA COM PASSO DE TEMPO FIXO	34
6.5	SIMULAÇÃO FÍSICA MULTI-THREAD	34
7	EXPERIMENTOS	36
7.1	AMBIENTE DE TESTE	36
7.2	CONFIGURAÇÃO DOS CENÁRIOS	36
7.2.1	Experimento 1 - Integrador de Jakobsen	36
7.2.2	Experimento 2 — Detecção de Colisões Convexas (SAT e GJK)	37
7.2.3	Experimento 3 — Simulação Multi-Threaded	37
7.3	MÉTRICAS DE AVALIAÇÃO	37
7.4	RESULTADOS E DISCUSSÃO	38
7.5	LIMITAÇÕES E TRABALHOS FUTUROS	38
8	CONCLUSÕES	39
	REFERÊNCIAS	40

1 INTRODUÇÃO

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

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

Em paralelo, o avanço dos **programas interativos**, especialmente jogos eletrônicos e simulações físicas em tempo real, intensificou a necessidade de técnicas capazes de combinar realismo visual com desempenho computacional. De acordo com Möller, Haines e Hoffman (2018) 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 imersiva.

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

1.1 CONTEXTUALIZAÇÃO DO PROBLEMA

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

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

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

1.2 MOTORES FÍSICOS E EVOLUÇÃO HISTÓRICA

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

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

1.3 APLICAÇÕES WEB E MOTIVAÇÃO TECNOLÓGICA

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

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

1.4 JUSTIFICATIVA

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

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

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

1.5 OBJETIVOS

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

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

1.6 ESTRUTURA DO TRABALHO

O Capítulo 2 apresenta os conceitos fundamentais da animação baseada em física, servindo como alicerce para a simulação. O Capítulo 3 detalha a abordagem central do trabalho, o método simplificado de Jakobsen, focando no integrador de Verlet e no relaxamento iterativo de restrições. O Capítulo 4 descreve os algoritmos essenciais para a detecção de colisões, como o SAT e o GJK. Em seguida, o Capítulo 5 explica as técnicas de resposta a colisão, incluindo a projeção de posição e o algoritmo EPA. O Capítulo 6 foca nas estratégias de otimização para garantir o desempenho em tempo real, cobrindo as fases Broad Phase e Narrow Phase, o uso de volumes delimitadores e o processamento

multi-thread. O Capítulo 7 apresenta a metodologia experimental, a avaliação do sistema e a discussão dos resultados obtidos. Por fim, o Capítulo 8 resume as contribuições do trabalho, discute limitações e propõe direções para trabalhos futuros.

2 ANIMAÇÃO BASEADA EM FÍSICA

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

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

2.1 CONCEITOS E DEFINIÇÕES

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

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

Segunda lei de Newton. Para um corpo de massa constante m submetido a uma força \vec{F} , o movimento é descrito por:

$$\vec{F} = m\vec{a} = m \frac{d\vec{v}}{dt} = m \frac{d^2\vec{x}}{dt^2}. \quad (2.1)$$

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

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

De maneira geral, um ciclo de simulação física engloba:

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

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

2.2 REPRESENTAÇÃO DE CORPOS RÍGIDOS

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

A representação matemática de um corpo rígido é dada por:

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

A dinâmica translacional e rotacional é governada pelas equações:

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

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

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

2.3 DINÂMICA DE PARTÍCULAS

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

O estado de uma partícula 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:

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

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

3 MÉTODO SIMPLIFICADO DE JAKOBSEN

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

3.1 MÉTODOS NUMÉRICOS EM SIMULAÇÃO

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

3.1.1 Método de Euler

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

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \quad (3.1)$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_t \Delta t \quad (3.2)$$

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

3.1.2 Método Semi-implícito de Euler

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

$$\vec{v}_{t+\Delta t} = \vec{v}_t + \vec{a}_t \Delta t \quad (3.3)$$

$$\vec{x}_{t+\Delta t} = \vec{x}_t + \vec{v}_{t+\Delta t} \Delta t \quad (3.4)$$

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

3.1.3 Integração de Verlet

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

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

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

3.2 RESTRIÇÕES GEOMÉTRICAS

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

3.2.1 Restrição Linear

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

$$|\vec{x}_i - \vec{x}_j| - d = 0 \quad (3.6)$$

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

Inserir figura

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

Algoritmo 1: Restrição Linear

$$\begin{aligned}\vec{\delta} &\leftarrow \vec{x}_2 - \vec{x}_1 \\ c &\leftarrow \frac{\|\vec{\delta}\| - d}{\|\vec{\delta}\|} \\ \vec{x}_1 &\leftarrow \vec{x}_1 - \epsilon \vec{\delta} c \\ \vec{x}_2 &\leftarrow \vec{x}_2 + \epsilon \vec{\delta} c\end{aligned}$$

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

3.2.2 Restrição de Revolução

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

(Inserir figura ilustrativa.)

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

3.2.3 Restrição Angular

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

$$|x_2 - x_1| > d$$

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

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

Algoritmo 2: Restrição Angular

$$\begin{aligned}\vec{\delta} &\leftarrow \vec{x}_2 - \vec{x}_1 \\ \text{if } |\vec{\delta}| < d \text{ then} \\ &\quad c \leftarrow \frac{|\vec{\delta}| - d}{|\vec{\delta}|} \\ &\quad x_1 \leftarrow x_1 - \vec{\delta} * \epsilon * c \\ &\quad x_2 \leftarrow x_2 + \vec{\delta} * \epsilon * c\end{aligned}$$

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

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

3.3 RESOLVENDO RESTRIÇÕES CONCORRENTES POR RELAXAMENTO

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

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

Algoritmo 3: Satisfazer Restrições

```

Input: n: número de repetições
for  $i \leftarrow 0$  to n do
    foreach restrição em restrições do
        SatisfazerRestricao(restrição)

```

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

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

4 DETECÇÃO DE COLISÕES

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.

4.1 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.

(INSERIR IMAGEM DE FORMA CONVEXA E NÃO-CONVEXA)

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 atravessasse 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.

4.2 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.*

INserir IMAGEM POLIGONO SEPARADO E POLIGONO EM COLISÃO

Um *plano de separação* (PS), definido por um vetor normal v e um escalar δ , é um hiperplano que separa A e B . Em termos computacionais, o SAT testa um conjunto finito de eixos candidatos — tipicamente as normais às arestas dos polígonos — verificando se existe algum eixo que sirva como separador.

4.2.1 Projeção sobre um eixo

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

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

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

De forma análoga, para o segundo polígono B :

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

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

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

4.2.2 Teste de interseção booleana

O pseudocódigo abaixo resume o teste booleano de interseção via SAT:

Algoritmo 4: Teste booleano de colisão via SAT

Input: Polígonos convexos A e B

Output: Verdadeiro se houver interseção

foreach aresta e de A e B **do**

$\hat{n} \leftarrow$ normal unitária de e	$p_i \leftarrow$ Projeção de A ao longo de \hat{n}
	$p_j \leftarrow$ Projeção de B ao longo de \hat{n}
	if p_i não sobrepõe p_j then
	return False

return True

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.

4.2.3 Vetor de Translação Mínima (MTV)

Para aplicações que exigem resposta física, é necessário calcular não apenas a detecção de colisão, mas também o deslocamento mínimo para separar os objetos. O MTV é obtido identificando-se o eixo cuja sobreposição projetada é mínima.

Algoritmo 5: SAT com cálculo do MTV

Input: Polígonos convexos A e B
Output: Direção v e penetração δ , ou falso
 $v \leftarrow 0$
 $\delta \leftarrow \infty$

```

foreach aresta  $e$  de  $A$  e  $B$  do
     $\hat{n} \leftarrow$  normal unitária de  $e$ 
     $p_1 \leftarrow$  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
         $\delta \leftarrow \Delta$ 
         $v \leftarrow \hat{n}$ 
return  $v, \delta$ 
```

4.3 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.

4.3.1 Soma de Minkowski

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

$$A + B = \{\vec{x} + \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \quad (4.5)$$

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

(INserir FIGURA SOMA DE MINKOWSKI)

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

$$A - B = \{\vec{x} - \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \quad (4.6)$$

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

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

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

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

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

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

4.3.2 Função de Suporte

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

A função de suporte deve retornar o ponto mais distante em uma direção dentro da Diferença de Minkowski. Isso cria um Simplex que contém uma área máxima, aumentando, portanto, a chance de que o algoritmo termine rapidamente. Além disso, podemos

usar o fato de que todos os pontos retornados desta forma estão na borda da Diferença Minkowski e, portanto, se não pudermos adicionar um ponto além da origem ao longo de alguma direção, sabemos que a Diferença de Minkowski não contém a origem. Isso aumenta as chances de o algoritmo sair rapidamente em casos de não interseção.

4.3.3 Construção iterativa do simplex

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

Algoritmo 6: GJK

Input: Polígonos convexos A e B

Output: Verdadeiro se ocorreu colisão

- 1 Escolhe-se uma direção inicial \vec{d} ;
 - 2 $p \leftarrow \text{support}(A \ominus B, \vec{d})$
 - 3 **if** $p \cdot \vec{d} < 0$ **then**
 - 4 **return** *False*
 - 5 Atualiza-se o simplex com p e calcula-se nova direção \vec{d} ;
 - 6 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.

5 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. Neste capítulo discutimos os princípios clássicos, as formulações modernas e a relação direta entre métodos geométricos empregados na fase de detecção — como o vetor de translação mínima (MTV) — e métodos baseados em partículas e restrições, como o modelo proposto por Jakobsen (2001).

5.1 FUNDAMENTOS DA RESPOSTA A COLISÃO

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.

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

5.2 MÉTODOS DINÂMICOS NA SIMULAÇÃO FÍSICA

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

5.2.1 Método do Impulso

Os métodos *Método do Impulso* tratam colisões aplicando impulsos instantâneos que alteram diretamente as velocidades dos corpos para preservar o momento linear e angular. Essa abordagem é 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.

5.2.2 Método de Penalidades

Nos métodos *Método 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.

5.2.3 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.

5.3 PROCESSO DE SEPARAÇÃO

A metodologia proposta por Jakobsen (2001), embora descrita em termos de integração Verlet e relaxamento de restrições geométricas, pode ser fundamentalmente entendida como uma precursora direta das modernas abordagens de *Position-Based Dynamics* (PBD), popularizadas por Müller et al. (2007).

Diferentemente de abordagens que atuam diretamente nas velocidades ou utilizam impulsos, 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: ponto de contato pertence a um vértice ou pertence a uma aresta.

INSERIR DIAGRAMA MOSTRANDO OS DOIS CASOS

Após a etapa de detecção de colisão, seja \vec{d} o *Minimum Translation Vector* (MTV), um vetor que representa a menor correção necessária para eliminar a interpenetração entre os objetos. O MTV é orientado do corpo A em direção ao corpo B .

A separação não é aplicada a um ponto rígido, mas distribuída entre as partículas que definem os corpos. Denotando por \vec{p} e \vec{q} os pontos de contato estimados em A e B , respectivamente, a correção por projeção é dividida igualmente entre os dois corpos:

$$\Delta\vec{p} = -\frac{1}{2}\vec{d}, \quad \Delta\vec{q} = +\frac{1}{2}\vec{d}. \quad (5.1)$$

Caso o ponto de contato seja um vértice único, basta mover os vértices para fora da região inválida. Caso seja uma colisão tipo aresta-vértice o ponto de contato 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

$$p = \alpha x_1 + (\alpha - 1)x_2, \quad 0 \leq \alpha \leq 1. \quad (5.2)$$

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. 5.2, 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

$$x_1+ = \alpha\Delta_p \quad (5.3)$$

$$x_2+ = (1 - \alpha)\Delta_p \quad (5.4)$$

$$\alpha = \frac{(\vec{p} - \vec{x}_2) \cdot (\vec{x}_1 - \vec{x}_2)}{\|\vec{x}_1 - \vec{x}_2\|^2}. \quad (5.5)$$

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.

5.4 ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA)

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

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

Este algoritmo é uma extensão porque sua entrada é o Simplex final do GJK que contém a origem e encontra o MTV. A distância entre o ponto mais próximo com a

origem é a profundidade de penetração (δ). Além disso, o vetor normal para o ponto mais próximo é a direção de separação (ponto de contato). A solução ingênuia é usar o normal da face mais próxima da origem, porém um simplex não precisa conter nenhuma das faces do polígono original, o quê pode acabar com uma normal incorreta.

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

Algoritmo 7: EPA

Input: Simplex

Output: separation v , penetration δ

```

for  $i \leftarrow 0$  to  $i < MAX\_ITERATION$  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  $|\delta - length(e)| < TOLERANCE$  then
         $\quad$  return  $normal(e), \delta$ 
    Adicionar ponto ao simplex

```

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

5.5 DESAFIOS

Como discutido anteriormente, o objetivo principal dessas técnicas não é fidelidade física absoluta, mas sim a geração de comportamentos verossímeis, estáveis e visualmente plausíveis. No entanto, essa busca por simplicidade e desempenho resulta em compromissos importantes.

5.5.1 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.

5.5.2 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.

5.5.3 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.

5.6 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.

6 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 $(n - 1) + (n - 2) + \dots + 1 = n(n - 1)/2 = O(n^2)$ testes de pares no pior caso. Devido à complexidade quadrática, testar ingenuamente cada par torna-se impraticável mesmo para valores moderados de n .

Reducir 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.

6.1 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 possibilitem 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.

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

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

6.1.1 Caixa Delimitadora Alinhada ao Eixo Coordenado (AABB)

A caixa delimitadora mínima para um conjunto de pontos em N dimensões é aquela que possui o menor volume possível e ainda assim contém todos os pontos. Notavelmente, a AABB mínima para um conjunto é a mesma que a AABB mínima de seu fecho convexo, fato útil em heurísticas para computação eficiente.

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 8: 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
     $\sqsubset$  return False
if  $A.y_{max} < B.y_{min}$  ou  $A.y_{min} > B.y_{max}$  then
     $\sqsubset$  return False
if  $A.z_{max} < B.z_{min}$  ou  $A.z_{min} > B.z_{max}$  then
     $\sqsubset$  return False
return True

```

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

6.1.2 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 :

$$OBB = \left\{ c + \sum_{i=1}^3 \alpha_i \hat{u}_i \mid -e_i \leq \alpha_i \leq e_i \right\} \quad (6.1)$$

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.

6.1.3 Esferas e Elipsoides

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

$$\text{Sphere} = \{x \in \mathbb{R}^3 \mid \|x - c\| \leq r\} \quad (6.2)$$

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).

6.1.4 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 9: 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.

6.2 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 10: Broad Phase generalizada

Output: Pares de objetos potenciais para colisão

vistos $\leftarrow \{\}$

pares_contato $\leftarrow \{\}$

foreach bodyA *em bodies* **do**

candidates \leftarrow 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 $\leftarrow \{\text{bodyA, bodyB}\}$

return pares_contato

6.2.1 Grade uniforme

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

6.3 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.

6.4 FÍSICA COM PASSO DE TEMPO FIXO

A forma mais ingênuas 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.

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

Algoritmo 11: Simulação física com passo variável

```

tempo_anterior ← agora()
while !sair do
    tempo_agora ← agora()
    dt ← tempo_agora - tempo_anterior
    tempo_anterior ← tempo_agora
    Física(dt)
    Renderizar()

```

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

Algoritmo 12: Simulação física com passo fixo

```

tempo_passado ← agora()
fixed dt ←  $\frac{1}{50}$ 
acumulador ← 0
while !sair do
    tempo_agora ← agora()
    dt ← tempo_agora - tempo_passado
    tempo_passado ← tempo_agora
    acumulador ← acumulador + dt
    while acumulador <= fixed dt do
        Física(dt)
        acumulador ← acumulador - fixed dt
    Física(dt)
    Renderizar()

```

6.5 SIMULAÇÃO FÍSICA MULTI-THREAD

Em programas interativos tradicionais, a lógica, a física e a renderização são executadas em um único *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.

7 EXPERIMENTOS

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

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

7.1 AMBIENTE DE TESTE

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

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

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

7.2 CONFIGURAÇÃO DOS CENÁRIOS

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

7.2.1 Experimento 1 - Integrador de Jakobsen

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

INserir IMAGENS

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

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

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

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

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

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

Resultados esperados:

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

7.2.3 Experimento 3 — Simulação Multi-Threaded

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

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

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

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

7.3 MÉTRICAS DE AVALIAÇÃO

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

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

7.4 RESULTADOS E DISCUSSÃO

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

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

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

INserir FIGURA

7.5 LIMITAÇÕES E TRABALHOS FUTUROS

Entre as limitações observadas, destacam-se:

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

Como trabalhos futuros, propõe-se:

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

8 CONCLUSÕES

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

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

REFERÊNCIAS

- AZEVEDO, A. C. E. **Computação gráfica - Teoria e prática.** [S.l.]: Editora Campus, Ltda, 2003.
- BARBER, C. B.; DOBKIN, D. P.; HUHDANPAA, H. The quickhull algorithm for convex hulls. **ACM Transactions on Mathematical Software (TOMS)**, Acm New York, NY, USA, v. 22, n. 4, p. 469–483, 1996.
- ERICSON, C. **Real-time collision detection.** [S.l.]: Crc Press, 2004.
- GILBERT, E.; JOHNSON, D.; KEERTHI, S. A fast procedure for computing the distance between complex objects in three-dimensional space. **IEEE Journal on Robotics and Automation**, v. 4, n. 2, p. 193–203, 1988.
- JAKOBSEN, T. Advanced character physics. In: **Proceedings of the Game Developer's Conference 2001**. San Jose, CA: Game Developers Conference, 2001. Presented at Game Developer's Conference 2001.
- MÖLLER, T.; HAINES, E.; HOFFMAN, N. **Real-time rendering.** 4. ed. [S.l.]: CRC Press, 2018.
- 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>.