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

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

RIO DE JANEIRO
2025

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

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

Orientador: Prof. Cláudio Esperança

RIO DE JANEIRO

2025

CIP - Catalogação na Publicação

R484t

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

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

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

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

DIEGO VASCONCELOS SCHARDOSIM DE MATOS

Animação Física Simplificada em Web

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

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Cláudio Esperança
Titulação (Instituição)

Nome do Professor1
Titulação (Instituição)

Nome do Professor2
Titulação (Instituição)

"The display is the computer."

Jen-Hsun Huang

RESUMO

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

Palavras-chave: Computação gráfica; Simulação 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

2 ANIMAÇÃO BASEADO EM FÍSICA 11
2.1 INTEGRADOR VERLET **11**
2.2 RESTRIÇÃO LINEAR **12**
2.3 RESTRIÇÃO DE REVOLUÇÃO **13**
2.4 RESTRIÇÃO ANGULAR **13**
2.5 RESOLVENDO RESTRIÇÕES CONCORRENTES POR RELAXAMENTO **14**

3 MODELANDO CORPOS 15
3.1 CORPOS RÍGIDOS **15**
3.2 CORPOS DEFORMÁVEIS **15**
3.3 CORPOS ARTICULADOS **15**
3.4 TECIDOS **16**

4 DETECÇÃO DE COLISÕES 17
4.1 POLÍGONOS CONVEXOS **17**
4.2 TEOREMA DO EIXO SEPARADOR (SAT) **17**
4.3 ALGORITMO GILBERT–JOHNSON–KEERTHI (GJK) **19**
4.3.1 Soma de Minkowski 19
4.3.2 Função Suporte 20
4.3.3 Iteração 21
4.4 POLÍGONOS CÔNCAVOS **21**
4.5 TRATANDO CASOS DEGENERADOS **21**

5 O FECHO CONVEXO 23
5.1 CAIXA DELIMITADORA ALINHADA AO EIXO COORDENADO (AABB) **24**
5.2 QUICKHULL **24**

6 RESPOSTA A COLISÃO 26
6.1 PROJEÇÃO DA POSIÇÃO PELO MÉTODO JAKOBSEN **26**
6.2 ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA) **26**

7 OTIMIZAÇÕES 28
7.1 BROAD PHASE **28**

7.1.1 Divisão espacial em grade uniforme 29
7.2 NARROW PHASE **29**
7.3 FÍSICA COM TAMANHO DE PASSO FIXO **29**
7.4 SIMULAÇÃO FÍSICA MULTI-THREAD **30**

8 EXPERIMENTOS 32
8.1 AMBIENTE DE TESTE **32**
8.2 CONFIGURAÇÃO DOS CENÁRIOS **32**
8.2.1 Experimento 1 - Integrador de Jakobsen 32
8.2.2 Experimento 2 — Detecção de Colisões Convexas (SAT e GJK) 33
8.2.3 Experimento 3 — Simulação Multi-Threaded 33
8.3 MÉTRICAS DE AVALIAÇÃO **33**
8.4 RESULTADOS E DISCUSSÃO **34**
8.5 LIMITAÇÕES E TRABALHOS FUTUROS **34**

9 CONCLUSÕES 35

REFERÊNCIAS 36

1 INTRODUÇÃO

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 (AZEVEDO, 2003).

A computação gráfica vista como ferramenta indicaria que temos um artista responsável pela arte gerada. Mesmo as imagens geradas a partir de equações podem ser consideradas arte, se essas equações forem fruto da criatividade e da capacidade do descobridor que manifesta sua habilidade e originalidade inventiva. A habilidade de simular a natureza em computadores tem sido objeto de atenção e curiosidade de toda a comunidade científica.

De acordo com Möller, Haines e Hoffman (2018) renderização em tempo real refere-se à criação rápida de imagens no computador. É a área mais interativa da computação gráfica. Uma imagem aparece na tela, o usuário interage ou reage, e esse feedback afeta o que será gerado em seguida.

Esse ciclo de reação e renderização acontece em uma velocidade 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). Com um quadro por segundo, há pouca sensação de interatividade; o usuário percebe claramente a chegada de cada nova imagem. A partir de cerca de 6 FPS, a sensação de interatividade começa a aumentar.

Uma taxa de quadros mais alta é importante para minimizar o tempo de resposta. Um atraso temporal de apenas 15 milissegundos pode prejudicar e interferir na interação. Como exemplo, os óculos de realidade virtual geralmente exigem 90 FPS para minimizar a latência.

Modelagem e animação baseada em física vêm sendo pesquisadas desde início desse século, encontrando aplicação em todas áreas de entretenimento, simulação, desenho assistido por computador e várias outras áreas.

Uma animação realista requer que os objetos em movimento obedeçam a leis físicas. Para tanto, vários aspectos precisam ser considerados como, por exemplo: no mundo real, dois objetos não podem ocupar o mesmo lugar no espaço ao mesmo tempo. Isto significa que objetos podem empurrar outros objetos dependendo de suas massas e velocidades; podem ser empilhados uns sobre os outros; não podem atravessar o chão, e assim por diante. Para incorporar um mecanismo que permita tratar estas situações uma tarefa importante para este mecanismo é detectar configurações onde haja interpenetração entre objetos, as quais são chamadas de "colisões".

Porém, sendo o processo de detecção de colisões geralmente muito custoso computacionalmente, a simulação de ambientes interativos dificilmente pode ser alcançada em tempo real. Isto se deve ao fato de que, a cada instante de tempo da simulação, diversas características físicas têm que ser computadas, tais como velocidades, forças, torques, momentos e outros.

O tratamento de colisões pode ser dividido em duas fases: na detecção de colisões, objetos que se interpenetram ou que estão em vias de o fazer são identificados, na resposta a colisões envolve a modificação dos diversos parâmetros físicos dos objetos envolvidos – tipicamente posição, orientação e velocidade – de tal forma que uma configuração fisicamente plausível seja obtida.

Há muitas abordagens para esse problema na literatura que envolve uso de métodos precisos que requerem computar diversas equações que regem as leis físicas, particularmente as da dinâmica como método do impulso, método da penalização

Jakobsen (2001) propôs um esquema simplificado de simulação baseada em física, conseguindo simular ambientes com objetos deformáveis e rígidos sem a necessidade de calcular explicitamente matrizes de orientação, torques ou tensores de inércia. Este esquema é baseado principalmente na implementação de restrições lineares e a com binação de diversas técnicas que se complementam:

- A dinâmica das partículas é simulada usando o integrador de Verlet.
- Esquema de resposta a colisão que consiste em projetar vértices que penetram uma determinada superfície para fora desta
- As restrições lineares são mantidas constantes usando um processo de relaxamento.
- É usada uma raiz quadrada aproximada em vez de uma exata para os cálculos de distância.
- Objetos (rígidos e deformáveis) são representados por sistemas de partículas com restrições lineares.

Entretanto, vários detalhes importantes foram suprimidos. Em particular, a ligação entre o mecanismo de detecção de colisão e os algoritmos de resposta a colisões é descrita apenas superficialmente, nenhuma estratégia é sugerida para tratar objetos que requeiram um número maior de restrições lineares e não é apresentado estruturas de otimizações para lidar com muitos objetos. Isto é relevante, já que o emprego de muitas restrições lineares pode comprometer o desempenho do sistema.

Neste trabalho estaremos focando em um protótipo de sistema baseado no Jakobsen apresentando soluções para detecção e resposta a colisão. Ele é adequado para situações de aplicações em tempo real e beneficia áreas que exigem alta imersão entre a simulação e o humano.

O restante deste trabalho é dividido nos seguintes capítulos: O Capítulo 2 apresenta as técnicas principais descritas por JAKOBSEN para animação baseada em física. No Capítulo 3 mostra as diversas representações de objetos aceitas pelos métodos anteriores. O Capítulo 4 aborda as principais metodologias para detecção de colisões e seus casos degenerados. O Capítulo 5 apresenta como calcular de forma simplificada as fronteiras dos objetos. No Capítulo 6 mostra como realizar a resposta a colisão por projeção das posições. O Capítulo 7 apresenta as otimizações usados neste trabalho. No Capítulo 8 os experimentos e comparação de resultados com outras bibliotecas. Finalmente, o Capítulo 9 aborda alguns comentários finais e sugestões para trabalhos futuros.

2 ANIMAÇÃO BASEADO EM FÍSICA

Na maioria dos casos, é preciso usar métodos sofisticados e exatos para simular a dinâmica. Porém, em aplicações de jogos interativos, a precisão não é realmente o mais importante, mas sim que o resultado final tenha uma aparência realista e que possa ser executado rapidamente.

Jakobsen (2001) apresentou um conjunto de métodos e técnicas que, unidas, conseguem atingir em grande parte estes objetivos. Estes métodos são relativamente simples de implementar (comparados com outros esquemas) e têm um bom desempenho. O algoritmo é iterativo e permite aumentar a precisão do método sacrificando parte de sua rapidez: se uma pequena fonte de imprecisão é aceita, o código pode conseguir uma execução mais rápida. Esta margem de erro ainda pode ser ajustada de forma adaptativa em tempo de execução.

Em geral, o sucesso deste método vem da combinação de varias técnicas que se beneficiam umas das outras, principalmente o uso do integrador Verlet, o tratamento de colisões usando projeção, e a resolução de restrições de distância usando relaxamento. A seguir serão descritas as componentes mais importantes da abordagem proposta por Jakobsen.

2.1 INTEGRADOR VERLET

O núcleo da simulação é um sistema de partículas, sua implementação mais simples é para cada passo calcular sua nova posição **x** e nova velocidade **v** com o sistema abaixo:

$$\begin{cases} x' = x + v \cdot \Delta t \\ v' = v + a \cdot \Delta t \end{cases}$$

Esse é o método de Euler sendo Δt o tamanho do passo, **a** é aceleração calculado a partir da lei de newton $F = ma$ onde F é a força acumulada agindo na partícula.

Existem muitos outros esquemas de integração como Runge-Kutta e outras variantes. Um método que é bem estável, sua velocidade é calculada implicitamente o quê mantém sua posição e velocidade em sincronia é o integrador de Verlet, muito popular em simulação de dinâmica molecular.

$$\begin{cases} x' = 2x - x^* + a \cdot \Delta t^2 \\ x^* = x \end{cases}$$

Sendo **x** sua posição corrente e x^* sua posição anterior, essa é chamada representação sem velocidade. Ao fim de cada passo conseguimos representar o sistema de partículas como:

$$X = \begin{pmatrix} x_1 \\ x_1^* \\ a_1 \\ \vdots \\ x_n \\ x_n^* \\ a_n \end{pmatrix}$$

O algoritmo para o integrador verlet segue como

Algoritmo 1: Verlet

Input: Tamanho do passo dt

for $i \leftarrow 0$ **to** n **do**

- $x \leftarrow X[i]$
- $x^* \leftarrow X[i+1]$
- $a \leftarrow X[i+2]$
- $x_{tmp} \leftarrow x$
- $X[i] \leftarrow 2 * x - x^* + a * dt$
- $X[i+1] \leftarrow x_{tmp}$
- $X[i+2] = 0$ `// Limpa as forças para próxima iteração`

Como dito anteriormente, as vantagens de usar esse esquema proposta por JAKOBSEN serão evidenciadas quando combinadas com as próximas técnicas.

2.2 RESTRIÇÃO LINEAR

O sucesso de Jakobsen (2001) não está relacionado apenas ao método numérico, mas foi conectar ideias simples para construir objetos maiores e complexos. Isso é feito atribuindo as partículas algo em comum, um conjunto de propriedades que devem ser respeitadas para atingir o efeito desejado.

A primeira propriedade - também é a base para as próximas - é chamada restrição linear ou restrição de distância. Ela apenas dita que duas partículas p_0 e p_1 devem sempre estar a uma distância d entre si. Dessa forma o conjunto de partículas devem satisfazer a todo instante uma coleção de inequações unilaterais representadas na forma:

$$|x_2 - x_1| = d \tag{2.1}$$

Mesmo que as posições das partículas estejam inicialmente corretas, depois de um passo de simulação a distância entre elas pode se tornar inválidas (através de forças externas, por exemplo). Para corrigir sua distância devemos mover (projetar) elas de tal forma que satisfaça 2.1.

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 2: Restrição Linear

$\vec{\delta} \leftarrow x_2 - x_1$
$\epsilon \leftarrow \frac{|\vec{\delta}|-d}{|\vec{\delta}|}$
$x_1 \leftarrow x_1 - \vec{\delta} * 0.5 * \epsilon$
$x_2 \leftarrow x_2 + \vec{\delta} * 0.5 * \epsilon$

O pseudocódigo 2 irá separar ou aproximar as partículas de tal forma que satisfaçam a distancia d. Essa situação é comparável a um sistema de molas interconectadas entre partículas de rigidez que tendem para o infinito ou a uma haste rígida separando as duas partículas.

2.3 RESTRIÇÃO DE REVOLUÇÃO

Na animação física queremos muitas vezes que uma partícula gire em torno de um eixo. Isso é feito de forma simples, basta ter uma partícula em comum cuja função seja ser um eixo de rotação.

Inserir figura

Uma outra forma de fazer isso é usar uma restrição linear com $d = 0$.

2.4 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

$$|x_2 - x_1| > 100$$

Com essa restrição conseguimos modelar propriedades como 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.

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

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

Algoritmo 3: Restrição Angular

$\vec{\delta} \leftarrow x_2 - x_1$
if $|\vec{\delta}| < d$ **then**
  $\epsilon \leftarrow \frac{|\vec{\delta}|-d}{|\vec{\delta}|}$
  $x_1 \leftarrow x_1 - \vec{\delta} * 0.5 * \epsilon$
  $x_2 \leftarrow x_2 + \vec{\delta} * 0.5 * \epsilon$

2.5 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, isso seria o equivalente a encontrar a solução de um sistema de equações.

Entretanto Jakobsen (2001) propõe uma abordagem de solução indireta por iteração local, basta satisfazer todas restrições e repetir um número N vezes a cada passo da simulação

Algoritmo 4: Satisfazer Restrições

Input: n: número de repetições
for $i \leftarrow 0$ **to** n **do**
  ResolverRestrição(i)

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 2 a constante 0,5 também 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.

3 MODELANDO CORPOS

Em animação física, convencionalmente são usadas funções paramétricas para representar sólidos simples, tais como cubos, cilindros, cones, esferas, toros, etc. Isto é feito frequentemente em ambientes interativos complexos, como jogos, onde é preciso animar milhares de objetos simples.

Um objeto geométrico é um conjunto fechado de pontos, limitado e não vazio. É fechado, pois a borda faz parte do objeto e é limitado significa que existe uma esfera de raio finito que limita o objeto. Assim, por exemplo, um plano é fechado mas não limitado. Tais objetos geométricos podem ser de dois tipos: objetos convexos e objetos côncavos. Um objeto é convexo se contém todos os segmentos de reta que conectam pares de pontos, caso contrário é chamado côncavo.

Objetos complexos podem ser compostos de objetos mais simples – tipicamente convexos.

3.1 CORPOS RÍGIDOS

Usando as ferramentas vistas no capítulo anterior a representação de corpos rígidos torna-se um passo quase natural a ser feito. Basta decompor a geometria desejada em partes menores e mais simples usando partículas para representar seus vértices e restrições lineares com coeficiente de restituição igual a 1 para representar suas arestas. No caso de 2D um quadrado pode ser representado com 4 partículas, 4 restrições lineares e mais 2 restrições internas para garantir rigidez.

Dessa forma ao colocar o objeto na simulação a integração Verlet das partículas e o relaxamento das restrições são responsáveis por mover o corpo de forma plausível, conservando momento e torque.

3.2 CORPOS DEFORMÁVEIS

A modelagem de um corpo deformável é análoga a um corpo rígido, com a principal diferença que o coeficiente de restituição é menor que 1. Dessa forma será necessário alguns passos da simulação para a restrição convergir para configuração ideal, gerando efeito de deformação do corpo.

3.3 CORPOS ARTICULADOS

Essa abordagem permite construir um modelo completo de corpo articulado bem realista usando uma combinação de: restrição linear, restrição de revolução e restrição angular.

3.4 TECIDOS

O tecido pode ser facilmente representado como uma grade uniforme de partículas conectadas por restrições lineares com coeficientes de restituição menor que 1 e apenas um passo de relaxamento.

4 DETECÇÃO DE COLISÕES

O mecanismo de detecção de colisão é um componente fundamental para um sistema de simulação baseada em física [26, 40, 29, 32, 33]. Este componente permite efetuar consultas rápidas de proximidade geométrica entre objetos durante a simulação. Isso não somente inclui verificações de como dois objetos se intersectam, mas também quando e onde a colisão ocorreu.

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

4.1 POLÍGONOS CONVEXOS

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

(INSERIR IMAGEM FORMA CONVEXA X NAO CONVEXA)

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

4.2 TEOREMA DO EIXO SEPARADOR (SAT)

O Teorema do Eixo Separador (do inglês Separating Axis Theorem, SAT) é um método para determinar se duas formas convexas estão se cruzando. O algoritmo também pode ser usado para encontrar o vetor de penetração mínimo que é útil para simulação de física e uma série de outras aplicações. O SAT é um algoritmo genérico rápido que pode remover a necessidade de ter código de detecção de colisão para cada par tipo de forma, reduzindo assim o código e a manutenção.

Teorema 1. *Se dois objetos convexos não estiverem penetrando, existe um eixo para o qual a projeção dos objetos não se sobreporá.*

Um plano de separação (PS) é um plano que está localizado entre dois objetos convexos, separando-os. Eventualmente, A está situado no lado positivo e B no lado negativo ou vice-versa. Matematicamente, um PS é definido por $H(v, \delta)$, onde v é chamado de eixo de separação (ES).

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

INSERIR IMAGEM POLIGONO SEPARADO E POLIGONO EM COLISÃO

Isso pode ser verificado calculando a linha perpendicular à linha que separa as duas formas e projetarmos as formas nessa linha. Caso haja sobreposição das projeções os objetos estão em interseção, caso contrário não há colisão. Uma linha onde as projeções (sombras) das formas não se sobrepõem é chamada de eixo de separação.

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

Algoritmo 5: SAT 1

```
Input: objetoA e objetoB
foreach eixo do objetoA do
    p1 ← Projetar objetoA no eixo
    p2 ← Projetar objetoB no eixo
    if p1 sobrepoe p2 then
        Result: False
foreach eixo do objetoB do
    p1 ← Projetar objetoA no eixo
    p2 ← Projetar objetoB no eixo
    if p1 sobrepoe p2 then
        Result: False
Result: True
```

O algoritmo SAT precisa responder duas perguntas fundamentais:

1. **Quais eixos de separação testar?** deve-se testar a normal de todas faces (3D) ou arestas (2D) dos objetos que estão participando da colisão.

2. **Como projetar um geometria no eixo?** Para projetar um polígono em um eixo é relativamente simples; loop sobre todos os vértices e faça o produto interno com o eixo, armazenando o mínimo e máximo.

Dessa forma o algoritmo apenas retorna verdadeiro ou falso se as duas formas estão se sobrepondo. Além disso, o SAT pode retornar um vetor de translação mínima (MTV). A MTV é o vetor usado para empurrar as formas para fora da colisão, na direção de separação com tamanho da penetração.

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

Algoritmo 6: SAT 2

```
Input: objetoA e objetoB
penetracao ← 0
separacao ← Vector.Zero
foreach eixo do objetoA do
    p1 ← Projetar objetoA no eixo
    p2 ← Projetar objetoB no eixo
    δ ← projeção p1 com p2
    if δ ≤ TOLERANCIA then
        Result: False
    else
        penetracao ← δ
        separacao ← eixo
foreach eixo do objetoB do
    p1 ← Projetar objetoA no eixo
    p2 ← Projetar objetoB no eixo
    δ ← projeção p1 com p2
    if δ ≤ TOLERANCIA then
        Result: False
    else
        penetracao ← δ
        separacao ← eixo
Result: penetracao, separacao
```

4.3 ALGORITMO GILBERT–JOHNSON–KEERTHI (GJK)

O algoritmo de distância Gilbert-Johnson-Keerthi (GJK) é um método de determinar a distância mínima entre dois conjuntos convexos, publicado pela primeira vez por GILBERT; JOHNSON; KEERTHI em 1988. Ao contrário de muitos outros algoritmos de distância, não requer que os dados de geometria sejam armazenados em qualquer formato específico, mas depende apenas de uma função de suporte para gerar iterativamente Simplex mais próximas da resposta correta usando a diferença de Minkowski.

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

4.3.1 Soma de Minkowski

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

$$A + B = \{\vec{x} + \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{4.1}$$

O objeto A + B é o conjunto de pontos obtido por um processo de varredura que

translada o centro de massa de B para cada ponto de A, ou seja, faz-se uma cópia do objeto B centrado em cada ponto de A

(INSERIR FIGURA SOMA DE MINKOWSKI)

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

$$A - B = \{\vec{x} - \vec{y} \mid \vec{x} \in A, \vec{y} \in B\}, \tag{4.2}$$

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

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

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

Teorema 2. *Se duas formas estiverem sobrepostas/em interseção a Diferença de Minkowski dessas formas conterá a origem.*

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 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 Iteração

Mesmo seguindo todos esses critérios ainda podemos não obter um Simplex que contém a origem nessas três etapas. Devemos criar iterativamente o Simplex de tal forma que o Simplex esteja se aproximando de conter a origem. Também precisamos verificar duas condições ao longo do caminho: 1) o simplex atual contém a origem? e 2) somos capazes de incluir a origem?

Algoritmo 7: GJK

Input: objeto A e objeto B
d ← B - A `// Direção inicial aponta para o centro de B`
simplex ← support(A, B, d)
d ← -d `// Inverte a direção para próxima iteração`
for $i \leftarrow 0$ **to** *MAX_INTERATION* **do**
  $a \leftarrow support(A, B, d)$
  if $a \cdot d \leq 0$ **then**
    return *false*
  simplex ← a
  if *simplex contém a origem* **then**
    return *true*
  d ← calcular próxima direção

Podemos determinar onde está a origem com relação ao simplex através da realização de uma série de testes em plano (testes de linha para 2D) onde cada teste consiste em produtos de pontos simples. O primeiro caso que deve ser tratado é o caso do segmento de linha. Podemos determinar se o simplex contém a origem examinando as regiões de Voronoi

4.4 POLÍGONOS CÔNCAVOS

4.5 TRATANDO CASOS DEGENERADOS

Algoritmo 8: Contém Origem

Input: simplex
Output: Boolean
if *|simplex| == 4* **then**
    `// Caso 3D`
    return *TestePoliedro()*
else if *|simplex| == 3* **then**
    `// Caso 2D`
    return *TesteTriangulo()*
else
    return *TesteLinha()*

5 O FECHO CONVEXO

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

Em 2 dimensões, temos um polígono, que pode ser descrito por uma lista ordenada de seus vértices. Em 3 dimensões, temos um poliedro, que precisa de estruturas um pouco mais complicadas para armazenar as faces que compõem a fronteira. Em dimensões maiores, as coisas se complicam ainda mais, pois precisaríamos de estruturas que representassem as faces de todas as dimensões.

INSERIR FIGURA COM EXEMPLOS DE USO

Testar diretamente a geometria de dois objetos para verificar se há colisão entre eles é frequentemente muito caro, especialmente quando os objetos consistem em centenas ou até milhares de polígonos. Para minimizar esse custo, o fecho convexo dos objetos geralmente são testados quanto à sobreposição antes que o teste de interseção geométrica seja realizado.

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

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

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

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

INSERIR IMAGENS COM TIPOS DIFERENTES FORMAS DE FECHO CONVEXO

A ideia principal por trás dos volumes delimitadores é preceder testes geométricos complexos com testes menos dispendiosos que permitem que o teste seja interrompido precocemente. Para suportar testes de sobreposição de baixo custo, o volume delimitador deve ter uma forma geométrica simples. Ao mesmo tempo, para tornar o teste de corte antecipado o mais eficaz possível, o volume delimitador também deve ser o mais ajustado possível, resultando em um trade-off entre o ajuste e o custo do teste de interseção.

5.1 CAIXA DELIMITADORA ALINHADA AO EIXO COORDENADO (AABB)

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

A caixa delimitadora alinhada aos eixos (em inglês "axis-aligned bounding box", ou AABB) é um dos volumes delimitadores mais comuns. É uma caixa retangular de seis lados (em 3D, quatro lados em 2D) sujeita à restrição de que as bordas da caixa devem ser paralelas aos eixos de coordenadas cartesianos.

INSERIR IMAGEM DE AABB

Algoritmo 9: Teste de Intersecção AABB

Input: Objeto A, Objeto B
Output: Boolean
if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  Result: False
if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  Result: False
if x_{max} *de* $A < x_{min}$ *de* B *ou* x_{min} *de* $A > x_{max}$ *de* B **then**
  Result: False
Result: True

5.2 QUICKHULL

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

O algoritmo de 2 dimensões pode ser dividido nas seguintes etapas:

Algoritmo 10: Quickhull 2D

Input: Polígono Convexo

Output: Lista dos vértices que representam o fecho convexo

1 Encontre os pontos com coordenadas mínimas e máximas x, pois estes sempre farão parte do casco convexo. Se muitos pontos com o mesmo mínimo/máximo x existirem, use os com o mínimo/máximo y, respectivamente.

2 Use a linha formada pelos dois pontos para dividir o conjunto em dois subconjuntos de pontos, que serão processados de forma recursiva. Em seguida, descrevemos como determinar a parte do casco acima da linha; a parte do casco abaixo da linha pode ser determinada de forma semelhante.

3 Determine o ponto acima da linha com a distância máxima da linha. Este ponto forma um triângulo com os dois pontos na linha.

4 Os pontos dentro desse triângulo não podem ser parte do casco convexo e, portanto, podem ser ignorados nos próximos passos.

5 Repita recursivamente os dois passos anteriores nas duas linhas formadas pelos dois novos lados do triângulo.

6 Continue até que não mais pontos sejam deixados, a recursão chegou ao fim e os pontos selecionados constituem o casco convexo.

Para dimensões maiores extrapola o escopo desse trabalho, consulte Barber, Dobkin e Huhdanpaa (1996) para melhores definições.

6 RESPOSTA A COLISÃO

Os chamados métodos baseados em penalidades tratam o contato inserindo molas nos pontos de penetração. Embora seja muito simples de implementar, esse método apresenta diversas desvantagens sérias. Por exemplo, é difícil escolher constantes de mola adequadas de forma que, por um lado, os objetos não penetrem demais e, por outro lado, o sistema resultante não se torne instável. Em outros métodos para simulação física, as colisões são tratadas retrocedendo o tempo (por meio de busca binária, por exemplo) até o ponto exato da colisão, tratando a colisão analiticamente a partir desse ponto e, em seguida, reiniciando a simulação. Isso não é muito prático do ponto de vista de tempo real, pois o código pode se tornar muito lento quando há muitas colisões.

Jakobsen (2001) propõe uma estratégia mais simples, os vértices que fazem parte da colisão são simplesmente projetados para fora do obstáculo. Por projeção, em termos gerais, entendemos mover o ponto o mínimo possível até que ele esteja livre do obstáculo. Normalmente, isso significa mover o ponto perpendicularmente para fora, em direção à superfície de colisão.

6.1 PROJEÇÃO DA POSIÇÃO PELO MÉTODO JAKOBSEN

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

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

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

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

6.2 ALGORITMO DE EXPANSÃO DE POLITOPOS (EPA)

Para realizar a separação de dois objetos usando o algoritmo SAT basta calcularmos o MTV como visto na seção 4.2. Já para o GJK é preciso fazer um segundo passo, uma

extensão do algoritmo que nos permite encontrar a normal correta e profundidade das colisões.

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

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

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

Algoritmo 11: EPA

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

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

7 OTIMIZAÇÕES

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

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

7.1 BROAD PHASE

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

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

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

Algoritmo 12: Broad Phase generalizada

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

7.1.1 Divisão espacial em grade uniforme

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

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

INSERIR IMAGEM DE DIVISÃO ESPACIAL EM GRANDE UNIFORME

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

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

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

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

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

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

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

7.2 NARROW PHASE

Essa etapa consiste em determinar as colisões exatas entre os pares dos subgrupos aceitos pela Broad Phase. Aqui deve-se usar o solver desejável como SAT ou GJK.

7.3 FÍSICA COM TAMANHO DE PASSO FIXO

A tamanho do passo da simulação física é tradicionalmente atribuída como a variação do tempo que o último quadro demorou para finalizar. Essa abordagem trás um passo

variável que será executada mais rápido ou mais lento dependendo do computador do usuário.

Algoritmo 13: Simulação física com passo variável

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

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

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

Algoritmo 14: 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()
```

Dessa forma a física da simulação fica controlável e ainda com um parâmetro ajustável que permite aumentarmos a precisão da simulação caso necessário e forma independente da máquina do usuário.

7.4 SIMULAÇÃO FÍSICA MULTI-THREAD

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

físicas que são renderizadas na tela.

INSERIR DIAGRAMA

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

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

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

INSERIR NOVO DIAGRAMA

Para resolver esse problema devemos executar a simulação física numa thread separada enquanto o thread principal apenas renderiza os dados de física calculados mais recentes. Dessa forma, a thread principal não precisa esperar a simulação física terminar para renderizar a configuração mais recente dos objetos. Mesmo em situações em que há uma carga pesada na renderização, o cálculo da física pode ser realizado em paralelo.

8 EXPERIMENTOS

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

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

8.1 AMBIENTE DE TESTE

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

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

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

8.2 CONFIGURAÇÃO DOS CENÁRIOS

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

8.2.1 Experimento 1 - Integrador de Jakobsen

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

INSERIR IMAGENS

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

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

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

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

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

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

Resultados esperados:

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

8.2.3 Experimento 3 — Simulação Multi-Threaded

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

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

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

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

8.3 MÉTRICAS DE AVALIAÇÃO

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

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

8.4 RESULTADOS E DISCUSSÃO

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

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

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

INSERIR FIGURA

8.5 LIMITAÇÕES E TRABALHOS FUTUROS

Entre as limitações observadas, destacam-se:

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

Como trabalhos futuros, propõe-se:

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

9 CONCLUSÕES

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

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

REFERÊNCIAS

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

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

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

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

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

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