



**COPPE/UFRJ**

## ANIMAÇÃO BASEADA EM FÍSICA COM SIMULAÇÃO DESACOPLADA

Jonas Fonseca Galante

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Claudio Esperança

Rio de Janeiro

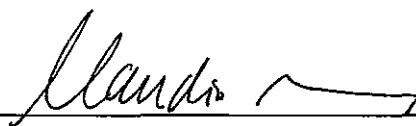
Junho de 2009

ANIMAÇÃO BASEADA EM FÍSICA COM SIMULAÇÃO DESACOPLADA

Jonas Fonseca Galante

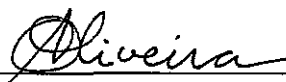
DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:



---

Prof. Claudio Esperança, Ph.D.



---

Prof. Antônio Alberto Fernandes de Oliveira, D.Sc.



---

Prof. João Luiz Dihl Comba, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

JUNHO DE 2009

Galante, Jonas Fonseca

Animação Baseada em Física com Simulação Desacoplada/Jonas Fonseca Galante. – Rio de Janeiro: UFRJ/COPPE, 2009.

XI, 69 p.: il.; 29, 7cm.

Orientador: Claudio Esperança

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2009.

Referências Bibliográficas: p. 64 – 69.

1. Detecção de colisão. 2. Corpos rígidos. 3. Animação física. I. Esperança, Claudio. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

# Agradecimentos

Aos professores do Laboratório de Computação Gráfica (LCG) da Universidade Federal do Rio de Janeiro (UFRJ), pela oportunidade de realizar o curso de mestrado e por transmitir os seus conhecimentos nas matérias cursadas.

Ao meu orientador, Claudio Esperança, pela paciência e ajuda com as idéias e conhecimentos passados que foram muito importantes.

Ao colega de LCG, Yalmar Ponce, pelos conselhos, idéias e por toda ajuda nesses anos.

À minha irmã, aos meus familiares e amigos, pelo incentivo, apoio e ajuda em todos os momentos.

E, principalmente, aos meus pais, Jorge e Glória Maria, por tudo que passaram e fizeram para que eu pudesse ter esta oportunidade na minha vida e pelos incentivos e apoio para superar os momentos difíceis.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

## ANIMAÇÃO BASEADA EM FÍSICA COM SIMULAÇÃO DESACOPLADA

Jonas Fonseca Galante

Junho/2009

Orientador: Claudio Esperança

Programa: Engenharia de Sistemas e Computação

Este trabalho descreve uma abordagem para a animação de corpos rígidos na qual o cálculo dos parâmetros físicos é desacoplado da exibição em si. Este processamento assíncrono torna possível tratar o cálculo físico com a precisão necessária, por exemplo, usando passos de integração variáveis. Adicionalmente, é possível armazenar quadros a serem exibidos futuramente de forma que um eventual atraso no cálculo da física não seja sentido pelo observador. É também apresentado um esquema de decomposição temporal da cena que viabiliza o aproveitamento de partes pré-computadas da animação mesmo em face da obsolescência de outras partes devido, por exemplo, a uma alteração interativa da cena. A abordagem proposta tende a produzir uma animação mais suave e com maior precisão. Um protótipo incorporando estas idéias foi implementado e experimentos foram realizados de forma a demonstrar a exequibilidade da proposta.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

## ANIMATION BASED ON PHYSICS WITH DECOUPLING SIMULATION

Jonas Fonseca Galante

June/2009

Advisor: Claudio Esperança

Department: Systems Engineering and Computer Science

This dissertation discusses a physically-based animation approach of rigid bodies in which the computation of physical parameters is decoupled from the proper exhibition. This asynchronous processing makes it possible to handle the physical computation with the necessary precision, using, for instance, variable integration time steps. Additionally, it is possible to store frames to be displayed in the future such that an eventual untimeliness in the physical computation may not be noticed by the observer. It is also presented a scheme for temporal decomposition of the scene which enables the use of parts of the pre-calculated frames even when other parts have to be discarded due to, say, an interaction. The proposed approach tends to produce a smoother and more precise animation. A prototype implementation of these ideas was used to conduct several experiments and thus show their feasibility.

# Sumário

<b>Lista de Figuras</b>	<b>ix</b>
<b>Lista de Tabelas</b>	<b>xi</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Animação Física</b>	<b>4</b>
2.1 Leis de Newton . . . . .	5
2.2 Dinâmica de Corpos Rígidos . . . . .	5
2.2.1 Posição e Orientação . . . . .	6
2.2.2 Velocidade Linear . . . . .	7
2.2.3 Velocidade Angular . . . . .	8
2.2.4 Massa de um corpo . . . . .	9
2.2.5 Centro de massa . . . . .	10
2.2.6 Força e Torque . . . . .	10
2.2.7 Momento Linear . . . . .	11
2.2.8 Momento Angular . . . . .	12
2.2.9 Tensor de Inércia . . . . .	13
2.2.10 Equações do Movimento de Corpos Rígidos . . . . .	14
2.3 Detecção de Colisões . . . . .	15
2.3.1 Volumes Limitantes . . . . .	15
2.3.2 Detecção de Colisão Simplificada . . . . .	20
2.3.3 Detecção de colisão exata . . . . .	21
2.4 Simulação de Corpos Rígidos . . . . .	27
2.4.1 Representação do Objeto . . . . .	27
2.4.2 Detecção de Interferência . . . . .	28

2.4.3	Integração . . . . .	29
2.4.4	Tratamento de Colisões . . . . .	30
2.4.5	Tratamento de Contatos . . . . .	32
2.4.6	Grafo de Contato . . . . .	33
2.4.7	Propagação de Choque . . . . .	33
2.4.8	Separação dos Objetos . . . . .	34
<b>3</b>	<b>Desacoplamento da Simulação</b>	<b>35</b>
3.1	Modelo tradicional . . . . .	36
3.2	Modelo Proposto . . . . .	39
<b>4</b>	<b>Implementação do Modelo Proposto</b>	<b>44</b>
4.1	Algoritmos . . . . .	47
<b>5</b>	<b>Resultados</b>	<b>53</b>
<b>6</b>	<b>Conclusões</b>	<b>62</b>
	<b>Referências Bibliográficas</b>	<b>64</b>



# Lista de Figuras

2.1	Relação do sistema de coordenadas do mundo com o sistema de coordenadas do corpo. . . . .	7
2.2	velocidade linear $v(t)$ e velocidade angular $\omega(t)$ de um corpo rígido. . . . .	8
2.3	O torque $\tau(t)$ devido à força $F_i(t)$ que age na partícula $r_i(t)$ do corpo rígido. . . . .	11
2.4	Esfera como volume limitante. . . . .	16
2.5	Duas esferas com interseção. . . . .	16
2.6	Duas esferas com interseção. . . . .	17
2.7	Caixas Limitantes de Orientação Arbitrária. . . . .	18
2.8	Caixa Limitante Alinhada com os Eixos. . . . .	19
2.9	Duas AABBs com interseção em apenas um dos eixos. . . . .	19
2.10	Método de detecção de colisão <i>sweep and prune</i> em 2D. . . . .	22
2.11	Hierarquia de caixas limitantes alinhadas com os eixos ( <i>AABB-Tree</i> ). . . . .	23
2.12	Hierarquia de esferas limitantes ( <i>Sphere-Tree</i> ). . . . .	23
2.13	Hierarquia de caixas limitantes orientadas ( <i>OBB-Tree</i> ). . . . .	23
2.14	Partículas representando a geometria de um objeto. . . . .	26
2.15	contatos entre objetos empilhados. . . . .	29
2.16	interferência entre objetos com arestas não proporcionais. . . . .	29
2.17	exemplo de objetos empilhados. . . . .	32
2.18	grafo de contato para cenas com objetos empilhados. . . . .	33
2.19	contatos entre objetos empilhados. . . . .	34
3.1	Exemplo da definição do $\Delta t$ . . . . .	36
3.2	Exemplo de uma situação ideal . . . . .	37
3.3	Exemplo de uma situação geral . . . . .	38

3.4	Demonstração da execução dos processos em paralelo . . . . .	39
3.5	As interações com a fila de estados . . . . .	40
3.6	Exemplos de interação com usuário. . . . .	42
4.1	Relação entre a física, a exibição e a fila de estados . . . . .	45
4.2	Exemplo de inserção de um objeto na cena . . . . .	51
5.1	Uma cena simples com 255 paralelepípedos representando dominós dispostos em espiral. . . . .	54
5.2	Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo do dominó. . . . .	54
5.3	Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo do dominó. . . . .	55
5.4	Exemplo com malhas complexas. . . . .	56
5.5	Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de malhas mais complexas. . . . .	56
5.6	Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo de malhas complexas. . . . .	57
5.7	Exemplo de dominós com interação do usuário. A barra vermelha representa o número mínimo de estados em fila e a barra azul indica o número máximo de estados em fila. . . . .	58
5.8	Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de dominós com interação do usuário. . . . .	58
5.9	Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo de dominós com interação do usuário. . . . .	59
5.10	Exemplo de uma pilha de paralelepípedos. . . . .	59
5.11	A pilha da esquerda sem retorno no tempo; A da direita com retorno no tempo. . . . .	60
5.12	Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de pilha de paralelepípedos . . . . .	60
5.13	Gráfico do tempo de exibição, da física e de $\Delta t$ utilizado em relação ao tempo real para o exemplo da pilha. . . . .	61

# Lista de Tabelas

# Capítulo 1

## Introdução

Animação física tem sido pesquisada há mais de uma década, podendo ser aplicada em filmes, desenhos, jogos digitais, engenharia, etc. Uma animação é a exibição de uma seqüência de imagens estáticas, chamadas de quadros (*frames*). Portanto, uma animação física consiste em uma simulação das leis da física sobre uma coleção de objetos, ilustrando o resultado através de animações. Também é comum possibilitar ao usuário interagir com objetos em tempo real, sendo que uma visualização em tempo real da cena é imprescindível, por exemplo, para os jogos digitais.

Atualmente, os jogos digitais conquistam público de todas as idades e estão presentes não apenas em consoles e computadores, mas também em celulares, dispositivos portáteis dedicados e até mesmo na TV digital interativa. Eles demandam cada vez mais recursos de hardware e exploram as possibilidades oferecidas pela Internet, simulando ambientes virtuais que permitem a interação síncrona com um grande número de jogadores. Devido a esta popularidade, os jogos digitais passaram a atrair grande interesse, sendo alvo de intensas pesquisas com objetivo de produzir jogos mais realistas. Conseqüentemente, novos conceitos foram incorporados na produção de jogos, como, por exemplo, o uso de motores de jogos (*game engines*).

Os motores de jogos são, atualmente, uma das peças fundamentais para o desenvolvimento de jogos, sendo responsáveis pela execução das funcionalidades de mais baixo nível necessárias durante a execução de um jogo. Em [1], são descritas várias características importantes que os motores de jogos devem atender. Tendo isto em vista, normalmente, apresentam-se alguns módulos já integrados, tais como o motor de física, responsável pelos cálculos das ações das leis da física nos objetos

dos jogos. A utilização deste módulo em jogos digitais ocasionou um maior interesse em pesquisas relacionadas à melhora do desempenho de simulações físicas. Atualmente, existem diversos motores de física com ótimos desempenhos, tais como *Bullet Physics Library* [2], *NVIDIA PhysX* [3] e outros.

A principal função do motor de física é garantir que dois objetos não ocupem o mesmo lugar no espaço ao mesmo tempo. Para assegurar este princípio da física, é importante detectar configurações onde haja interpenetração entre objetos, as quais são chamadas de colisões, e tratar estas colisões. A partir disto, objetos podem empurrar outros objetos dependendo de suas massas e velocidades, ser empilhados uns sobre os outros, e assim por diante. Portanto, a aplicação das leis da física nas animações permite a produção de efeitos realistas nos movimentos dos objetos e tratamento automático das colisões. No entanto, os cálculos numéricos necessários para produzir estes movimentos eficientemente são complexos e custosos computacionalmente, devido, principalmente, à complexidade da geometria dos objetos. Assim, a simulação de ambientes interativos dificilmente pode ser alcançada em tempo real.

Normalmente, o tratamento de colisões pode ser dividido nas fases de detecção de colisões e de resposta a colisões. Na detecção de colisões, são identificados os objetos interpenetrados ou os que estão próximos disto, enquanto que a resposta a colisões consiste na modificação dos diversos parâmetros físicos dos objetos envolvidos - tipicamente posição, orientação e velocidade - de tal forma que uma configuração fisicamente plausível seja obtida.

Em geral, animação física demanda uma quantidade considerável de recursos computacionais. Isto é necessário porque a cada instante de tempo da simulação, diversas características físicas devem ser computadas, tais como velocidades, forças, torques, momentos e outras.

Diversas abordagens [4, 5, 6, 7] para vários problemas associados à animação física podem ser encontradas na literatura. A maior parte dessas técnicas se baseia numa abordagem tradicional e requerem, em geral, que se usem métodos precisos para computar as diversas equações que regem as leis físicas, particularmente as da dinâmica. Contudo, em alguns campos de aplicação tais como jogos digitais, por exemplo, a precisão pode ser sacrificada em prol de uma maior velocidade de

simulação.

Normalmente, o modelo de processamento tradicionalmente implementado para a animação física é seqüencial. Assim, a exibição de um quadro da animação física somente é feita após o término do cálculo dos parâmetros físicos dos objetos. Os tempos de cálculo variam de acordo com a complexidade dos objetos envolvidos e da configuração da cena. Portanto, nos casos de cenas complexas, a animação fica parada enquanto aguarda a finalização do quadro seguinte.

Identificada essa limitação, este trabalho descreve um modelo para a animação física na qual o cálculo dos parâmetros físicos é desacoplado da exibição em si. A idéia é permitir o cálculo de cada quadro da animação com passos de tempo variados e não determinados pela exibição na tela. Assim, aproveita-se o tempo entre a exibição de cada quadro para calcular quadros a serem exibidos em instantes de tempo futuros.

Os próximos capítulos deste trabalho são divididos da seguinte forma: O Capítulo 2 faz um apanhado geral dos conceitos teóricos associados à animação física. O Capítulo 3 aborda as características do modelo proposto neste trabalho. O Capítulo 4 apresenta alguns resultados comparativos entre o modelo tradicional e o proposto. Finalmente, o Capítulo 5 apresenta alguns comentários finais e sugestões para trabalhos futuros.

# Capítulo 2

## Animação Física

Com a popularidade da animação física, diversas abordagens têm sido propostas [4, 6, 7, 8, 9, 10] na literatura. Tendo isto em vista, observa-se que uma das principais características da animação física é a representação dos objetos no tempo. Portanto, as posições, as orientações e as velocidades podem ser descritas em função do tempo.

Como mencionado anteriormente, a animação física é uma simulação das leis da física sobre uma coleção de objetos. Estes objetos podem ser de dois tipos: rígidos ou deformáveis. Apesar da separação em dois tipos, as alterações na localização dos objetos são feitas considerando todos os objetos como um corpo rígido. Estas alterações na localização compreendem essencialmente duas transformações: translações e rotações. Para tanto, considera-se que o objeto é modelado em um sistema próprio de coordenadas. Assim, a localização de um objeto no tempo  $t$  é representada por  $x(t)$  e  $R(t)$ , respectivamente, a posição e a orientação do objeto no sistema de coordenadas do mundo. Objetos deformáveis são usualmente representados por malhas poligonais, onde as posições dos vértices dos polígonos são dependentes do tempo. Assim, as deformações podem ser usadas para simular, por exemplo, fluidos, tecidos, ou pele.

Assim, pode-se dizer também que a animação física consiste num processo pelo qual os movimentos dos objetos são representados a partir da simulação das leis da física. Para tanto, propriedades como massa e densidade são atribuídas aos objetos, de forma a serem utilizadas no cálculo de propriedades dinâmicas tais como posição, orientação, velocidade e aceleração. Estes cálculos são estudados no item 2.2.

## 2.1 Leis de Newton

A dinâmica é a parte da física que estuda o movimento dos corpos. Tem como base as três leis de Newton (leis do movimento):

**Primeira lei de Newton ou Lei da Inércia:** Todo corpo permanece no estado de repouso ou em movimento retilíneo uniforme, a menos que a ele sejam aplicadas forças externas.

**Segunda lei de Newton ou Lei Fundamental da Mecânica/Dinâmica:**

Todo corpo precisa de uma força para se movimentar ou de uma força para parar. Portanto, o corpo adquire a velocidade e o sentido da resultante das forças aplicadas. Assim, quanto mais intensa for a força resultante, maior será a aceleração adquirida pelo corpo. Considerando-se um corpo de massa constante  $m$  com uma força  $F$  aplicada, o movimento do corpo em relação ao tempo é dado por  $\vec{F} = m \times \vec{a} = m \times \dot{v} = m \times \ddot{x}$ .

**Terceira lei de Newton ou Lei da Ação e Reação:** Se um corpo  $A$  aplicar uma força sobre um corpo  $B$ , receberá deste uma força de mesma intensidade, mesma direção e sentido oposto à força que aplicou em  $B$ . Portanto, a força que  $A$  exerce em  $B$  ( $\vec{F}_{AB}$ ) e a correspondente força que  $B$  exerce em  $A$  ( $\vec{F}_{BA}$ ) constituem o par ação-reação desta interação de contato (colisão). De acordo com esta lei, a relação das forças é  $\vec{F}_{AB} = -\vec{F}_{BA}$ .

Assim, para simular o movimento é necessário que as variáveis de estado dos objetos estejam relacionadas. Por exemplo, a posição de um objeto  $x(t)$  está relacionada com a velocidade pela equação  $v(t) = \dot{x}(t)$ , e com a aceleração pela equação  $a(t) = \ddot{x}(t)$ . Pode-se observar que a velocidade  $v(t)$  e a aceleração  $a(t)$  são dadas por equações diferenciais. Na prática, a simulação do movimento dos objetos requer o uso de métodos numéricos que resolvam estas equações diferenciais. Na seção 2.4.3, é descrito o método usado neste trabalho.

## 2.2 Dinâmica de Corpos Rígidos

Corpos rígidos são representados frequentemente por sistemas de partículas. Para simular o movimento de uma partícula, é preciso conhecer a força que age sobre



ela no instante  $t$ . Seja  $F(t)$  a força resultante que age na partícula no tempo  $t$ . A função  $F(t)$  é a soma de todas as forças que agem na partícula: gravidade, vento, forças de mola, etc. Se a partícula tem massa  $m$ , então a variação de  $X(t)$  ao longo do tempo é dada por

$$\frac{d}{dt}X(t) = \frac{d}{dt} \begin{pmatrix} x(t) \\ v(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \frac{F(t)}{m} \end{pmatrix}. \quad (2.1)$$

Dado um valor de  $X(t)$ , a equação 2.1 descreve como  $X(t)$  é instantaneamente alterado no tempo  $t$ .

Contudo, corpos rígidos diferem de partículas visto que ocupam um lugar no espaço e podem sofrer rotações. Portanto, a dinâmica de corpos rígidos requer considerações adicionais. Em suma, para simular o movimento de um corpo rígido é necessário uma equação similar à usada para simular o movimento de uma partícula (eq. 2.1), onde é necessário determinar  $\frac{d}{dt}X(t)$ , conforme é explicado abaixo.

### 2.2.1 Posição e Orientação

A posição de uma partícula no espaço no instante  $t$  pode ser representada pelo vetor  $x(t)$ , o qual descreve o deslocamento em relação à origem. De modo similar, a posição de um corpo no instante  $t$  é descrito por um vetor  $x(t)$ , que descreve o deslocamento do corpo em relação à origem. Adicionalmente, um corpo rígido pode sofrer rotações. Uma rotação pode ser representada por diferentes técnicas tais como matrizes, ângulos de Euler, ou quatérnios. Se usar matrizes, então a rotação de um corpo rígido pode ser representada diretamente por uma matriz  $R(t)$ . Então,  $x(t)$  e  $R(t)$  são as variáveis que descrevem o estado do corpo rígido.

Devido ao fato de que um corpo pode sofrer apenas translações e rotações, define-se a forma de um objeto em termos de um espaço fixo e imutável chamado *espaço do corpo*. Dada uma descrição geométrica do corpo nesse espaço, usa-se  $x(t)$  e  $R(t)$  para transformar o espaço do corpo no espaço do mundo, como pode ser observado na figura 2.1. Para simplificar algumas equações, é conveniente que o centro de massa do corpo coincida com  $O'$ , a origem do espaço do corpo.

Como  $R(t)$  representa a rotação do corpo em torno do centro de massa, então um vetor  $r$  fixo no espaço do corpo será transformado no vetor  $R(t) \cdot r$  no espaço

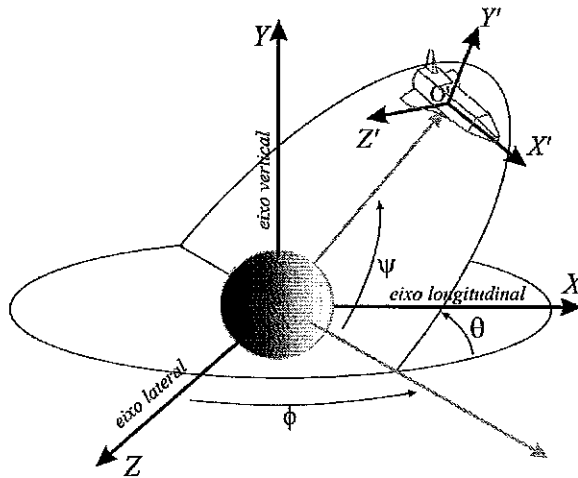


Figura 2.1: Relação do sistema de coordenadas do mundo com o sistema de coordenadas do corpo.

do mundo para o tempo  $t$ . De modo similar, se  $p_0$  é um ponto arbitrário do corpo rígido no espaço do corpo, então  $p(t)$ , sua posição no espaço do mundo, é obtida aplicando a rotação seguida da translação, ou seja:

$$p(t) = R(t) \cdot p_0 + x(t). \quad (2.2)$$

Assumindo que o centro de massa do corpo está localizado na origem do sistema de coordenadas do corpo, a posição do centro de massa no espaço do mundo é dada diretamente por  $x(t)$ . Isto significa que  $x(t)$  é a localização do centro de massa no espaço do mundo no tempo  $t$ .

### 2.2.2 Velocidade Linear

A velocidade linear descreve a mudança da posição do centro de massa do corpo no tempo. Assim, define-se  $x(t)$  como a *posição* do corpo no tempo  $t$ . Então,  $\dot{x}(t)$  é a velocidade do centro de massa no espaço do mundo e é definida como:

$$v(t) = \dot{x}(t) = \frac{d}{dt}x(t). \quad (2.3)$$

Imaginando que a orientação do corpo é fixa, então o corpo experimentará apenas uma translação pura. A quantidade vetorial  $v(t)$  expressa a velocidade desta translação (ver a figura 2.2).

### 2.2.3 Velocidade Angular

Além da translação, um corpo rígido também pode girar ao redor de um eixo fixo que passa por sua origem. Imagina-se que a posição do centro de massa está fixa no espaço do mundo. Então, qualquer movimento dos pontos do corpo só poderá ocorrer mediante uma rotação em torno de algum eixo que passe pelo centro de massa. Comumente, esta rotação é denotada pelo vetor  $w(t)$  e é conhecida como *velocidade angular*. A direção de  $w(t)$  coincide com a direção do eixo de rotação do corpo (ver a figura 2.2).

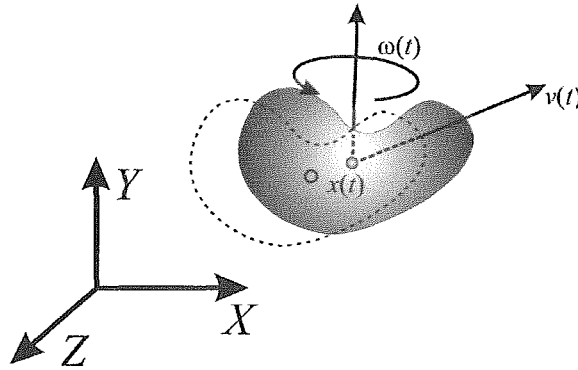


Figura 2.2: velocidade linear  $v(t)$  e velocidade angular  $\omega(t)$  de um corpo rígido.

A magnitude de  $w(t)$ , denotada por  $|w(t)|$ , nos diz quão rápida é a rotação. A velocidade linear relaciona-se com a posição do corpo através da equação 2.3. Analogamente, a rotação  $R(t)$  e a velocidade angular  $w(t)$  estão relacionadas. Porém,  $\dot{R}(t)$  não é  $w(t)$ , já que  $R(t)$  é uma matriz e  $w(t)$  é um vetor. Por outro lado, as colunas de  $R(t)$  representam os eixos  $x$ ,  $y$ , e  $z$  no tempo  $t$ ; isto significa que as colunas de  $\dot{R}(t)$  descrevem a velocidade com que os eixos  $x$ ,  $y$ , e  $z$  são transformados. Então de acordo com Witkin et al. [11] pode-se escrever:

$$\dot{R}(t) = \left( w(t) \times \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \quad w(t) \times \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \quad w(t) \times \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right). \quad (2.4)$$

Essa expressão é bastante complicada, mas pode ser reescrita de uma maneira mais compreensível. Existe uma característica dos vetores no espaço tridimensional que permite obter um resultado interessante. Dado um vetor  $v$ , se define  $v^*$  como a

matriz anti-simétrica de  $v$  dada por

$$\begin{pmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{pmatrix}, \quad (2.5)$$

o que permite definir  $u^* \cdot v = u \times v$ , e esta igualdade pode ser usada para reescrever a equação (2.4) como

$$\dot{R}(t) = \left( w(t)^* \cdot \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} w(t)^* \cdot \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} w(t)^* \cdot \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right). \quad (2.6)$$

Segundo as regras básicas de multiplicação, pode-se fatorar na seguinte expressão

$$\dot{R}(t) = w(t)^* \cdot \left( \begin{pmatrix} r_{xx} \\ r_{xy} \\ r_{xz} \end{pmatrix} \begin{pmatrix} r_{yx} \\ r_{yy} \\ r_{yz} \end{pmatrix} \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix} \right), \quad (2.7)$$

que é nada mais do que uma multiplicação de matrizes. Finalmente  $\dot{R}(t)$  é expresso como

$$\dot{R}(t) = w(t)^* \cdot R(t). \quad (2.8)$$

Isto, nos dá uma relação entre  $\dot{R}(t)$  e  $w(t)$ . Analogamente, pode-se relacionar um vetor  $r(t)$  com a velocidade angular  $w(t)$  através da equação  $\dot{r}(t) = w(t) \times r(t)$ .

## 2.2.4 Massa de um corpo

Para trabalhar com animação baseada em física, é preciso efetuar alguns cálculos de integração sobre o volume dos corpos. Para tornar estes cálculos mais simples, um corpo frequentemente é representado por um sistema de partículas. As partículas são indexadas de 1 até  $N$ , onde a  $i$ -ésima partícula tem massa  $m_i$ , e tem posição constante  $r_{0i}$  no espaço do corpo. A localização da  $i$ -ésima partícula no espaço do mundo no tempo  $t$  é denotada por  $r_i(t)$ , e está definida como

$$r_i(t) = R(t)r_{0i} + x(t). \quad (2.9)$$

A massa total do corpo, a qual é denotada por  $M$ , é a soma das massas das partículas

$$M = \sum_{i=1}^N m_i. \quad (2.10)$$

### 2.2.5 Centro de massa

O centro de massa de um corpo é definido por

$$O_c = \sum_{i=1}^N \frac{m_i r_i(t)}{M} = x(t), \quad (2.11)$$

onde  $M$  é a massa do corpo. Anteriormente foi definido que  $x(t)$  é a localização do centro de massa no espaço do mundo no tempo  $t$ . Isto pode ser demonstrado observando que, como a  $i$ -ésima partícula tem posição  $r_i(t) = R(t)r_{0i} + x(t)$  no espaço do mundo no tempo  $t$ , então o centro de massa no tempo  $t$  é dado por

$$\begin{aligned} \sum_{i=1}^N \frac{m_i r_i(t)}{M} &= \sum_{i=1}^N \frac{m_i (R(t)r_{0i} + x(t))}{M} \\ &= \frac{\sum_{i=1}^N m_i R(t)r_{0i} + \sum_{i=1}^N m_i x(t)}{M} \\ &= \frac{x(t) \sum_{i=1}^N m_i}{M} \\ &= x(t). \end{aligned} \quad (2.12)$$

### 2.2.6 Força e Torque

Ao representar um corpo rígido como um sistema de partículas, é conveniente imaginar que todas as forças que agem sobre o corpo são aplicadas a cada partícula individualmente.

$$\tau_i(t) = (r_i(t) - x(t)) \times F_i(t). \quad (2.13)$$

Torques diferem de forças, visto que o torque na partícula depende da localização  $r_i(t)$  da partícula, relativa ao centro de massa  $x(t)$ . De maneira intuitiva, pode-se pensar em  $\tau_i(t)$  como o eixo de rotação do corpo por influência da força  $F_i(t)$ , assumindo que o centro de massa está firmemente localizado (ver a figura 2.3).

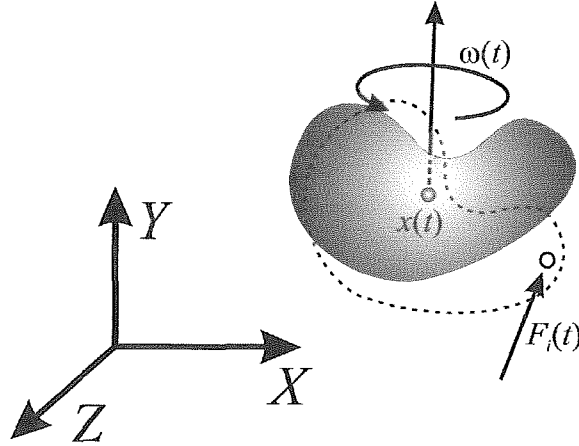


Figura 2.3: O torque  $\tau(t)$  devido à força  $F_i(t)$  que age na partícula  $r_i(t)$  do corpo rígido.

A força externa resultante  $F(t)$  que age no corpo é o somatório de  $F_i(t)$ ,

$$F(t) = \sum F_i(t). \quad (2.14)$$

Analogamente, o torque externo resultante é definido por

$$\tau(t) = \sum \tau_i(t) = \sum (r_i(t) - x(t)) \times F_i(t). \quad (2.15)$$

### 2.2.7 Momento Linear

O momento linear  $\rho$  de uma partícula de massa  $m$  e velocidade  $v$  é definido como  $\rho = mv$ . De maneira análoga, o momento linear  $P(t)$  de um corpo rígido é definido como

$$P(t) = \sum m_i \dot{r}_i(t), \quad (2.16)$$

onde a velocidade  $\dot{r}_i(t)$  da  $i$ -ésima partícula é  $\dot{r}_i(t) = v(t) + w(t) \times (r_i(t) - x(t))$ . Assim, o momento linear resultante do corpo é definido por

$$\begin{aligned}
P(t) &= \sum m_i \dot{r}_i(t) \\
&= \sum (m_i v(t) + m_i w(t) \times (r_i(t) - x(t))) \\
&= \sum m_i v(t) + w(t) \times \sum m_i (r_i(t) - x(t)) \\
&= \sum m_i v(t) = Mv(t).
\end{aligned} \tag{2.17}$$

Isto nos dá uma expressão simples para o momento linear resultante de um corpo. Se o corpo é uma única partícula de massa  $M$  e velocidade  $v(t)$ , então obtém-se uma relação importante, derivando a equação 2.17 e rearranjando os termos:

$$\dot{v}(t) = \frac{\dot{P}(t)}{M}. \tag{2.18}$$

Finalmente, a força resultante também pode ser obtida derivando-se o momento linear resultante  $\dot{P}(t) = F(t)$ .

## 2.2.8 Momento Angular

Apesar do significado de momento linear ser bastante intuitivo, o conceito de momento angular para um corpo rígido não é tão simples. Para o momento linear, tem-se que  $P(t) = Mv(t)$ . Analogamente, o momento angular resultante  $L(t)$  de um corpo rígido é dado por  $L(t) = I(t)w(t)$ . Aqui aparece termo  $I(t)$ , chamado de *tensor de inércia*, que corresponde a uma matriz  $3 \times 3$ . O tensor de inércia  $I(t)$  descreve como a massa do corpo está distribuída em relação a seu centro de massa. Note que em ambos os casos linear e angular, o momento é uma função linear da velocidade, mas enquanto que para o momento angular o fator de escala é uma matriz, para o momento linear, é um escalar. Note-se também que  $L(t)$  é independente de translações e, de modo similar,  $P(t)$  é independente de rotações. A relação entre  $L(t)$  e o torque resultante  $\tau(t)$  é simplesmente a derivada do momento angular  $L(t)$ , isto é,  $\dot{L}(t) = \tau(t)$ , o que é análogo à relação entre  $P(t)$  e  $F(t)$ .

## 2.2.9 Tensor de Inércia

O tensor de inércia  $I(t)$  é o fator de escala entre o momento angular  $L(t)$  e a velocidade angular  $w(t)$ . Dado um instante  $t$ , seja  $r'_i(t)$  o deslocamento da  $i$ -ésima partícula em relação a  $x(t)$ , isto é,  $r'_i(t) = r_i(t) - x(t)$ . O tensor de inércia  $I(t)$  é expresso em termos de  $r'_i(t)$  como a matriz simétrica

$$I(t) = \sum \begin{pmatrix} m_i(r'_{iy}{}^2 + r'_{iz}{}^2) & -m_i r'_{ix} r'_{iy} & -m_i r'_{ix} r'_{iz} \\ -m_i r'_{iy} r'_{ix} & m_i(r'_{ix}{}^2 + r'_{iz}{}^2) & -m_i r'_{iy} r'_{iz} \\ -m_i r'_{iz} r'_{ix} & -m_i r'_{iz} r'_{iy} & m_i(r'_{iy}{}^2 + r'_{iz}{}^2) \end{pmatrix}. \quad (2.19)$$

À primeira vista, parece necessário computar os somatórios para achar  $I(t)$  toda vez que há uma mudança na orientação do corpo. Isto pode ser custoso durante a simulação, a menos que os objetos tenham formas simples (por exemplo, esferas, cubos, etc). Entretanto, é possível computar o tensor de inércia a um baixo custo pré-computando estes somatórios em coordenadas do espaço do corpo. Usando o fato de que  $r_i'^T r'_i = r'_{ix}{}^2 + r'_{iy}{}^2 + r'_{iz}{}^2$ , pode-se re-escrever  $I(t)$  como a diferença

$$I(t) = \sum m_i r_i'^T r'_i \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} - \begin{pmatrix} m_i r'_{ix}{}^2 & m_i r'_{ix} r'_{iy} & m_i r'_{ix} r'_{iz} \\ m_i r'_{iy} r'_{ix} & m_i r'_{iy}{}^2 & m_i r'_{iy} r'_{iz} \\ m_i r'_{iz} r'_{ix} & m_i r'_{iz} r'_{iy} & m_i r'_{iz}{}^2 \end{pmatrix}. \quad (2.20)$$

Se denotar a matriz identidade  $3 \times 3$  por  $\mathbf{1}$ ,  $I(t)$  pode ser expresso como

$$I(t) = \sum m_i (r_i'^T r'_i) \mathbf{1} - r_i' r_i'^T, \quad (2.21)$$

como  $r_i(t) = R(t)r_{0i} + x(t)$  onde  $r_{0i}$  é uma constante, tem-se  $r'_i = R(t)r_{0i}$ . Visto que  $R(t)R(t)^T = \mathbf{1}$ , então pode-se re-escrever  $I(t)$  como



$$\begin{aligned}
I(t) &= \sum m_i (r_i'^T r_i') \mathbf{1} - r_i' r_i'^T \\
&= \sum m_i ((R(t)r_{0i})^T (R(t)r_{0i}) \mathbf{1} - (R(t)r_{0i})(R(t)r_{0i})^T) \\
&= \sum m_i ((r_{0i}^T r_{0i}) \mathbf{1} - R(t)r_{0i} r_{0i}^T R(t)^T) \\
&= \sum m_i (R(t)(r_{0i}^T r_{0i}) R(t)^T \mathbf{1} - R(t)r_{0i} r_{0i}^T R(t)^T) \\
&= R(t) (\sum m_i ((r_{0i}^T r_{0i}) \mathbf{1} - r_{0i} r_{0i}^T) R(t)^T). \tag{2.22}
\end{aligned}$$

Já que  $r_{0i} r_{0i}^T$  é um escalar, e usando  $I_b = \sum m_i ((r_{0i}^T r_{0i}) \mathbf{1} - r_{0i} r_{0i}^T)$ , o tensor de inércia é expresso como

$$I(t) = R(t) I_b R(t)^T, \tag{2.23}$$

onde pode-se observar que  $I_b$  é especificado no espaço do corpo.

### 2.2.10 Equações do Movimento de Corpos Rígidos

No começo desta seção foram apresentadas as equações do movimento para uma partícula simples, onde seu estado é representado por um único vetor  $X(t)$  que reúne as componentes  $x(t)$  e  $v(t)$ . De modo similar, um corpo rígido pode ser representado por um vetor  $X(t)$  expresso por

$$X(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix}, \tag{2.24}$$

e as demais características são obtidas derivando-se  $X(t)$

$$\frac{d}{dt} X(t) = \frac{d}{dt} \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ w(t)^* \cdot R(t) \\ F(t) \\ \tau(t) \end{pmatrix}. \tag{2.25}$$

Esta formulação é a tradicionalmente usada na maioria das abordagens para

animação baseada em física.

## 2.3 Detecção de Colisões

Como mencionado anteriormente, a detecção de colisão é um componente fundamental em sistemas com animação física. Normalmente, é um processo lento uma vez que qualquer par de objetos pode, em princípio, estar em colisão, ou seja, é um problema com grau de complexidade proporcional à complexidade da geometria envolvida. Além disso, como o tempo na computação gráfica é discretizado e a maioria dos algoritmos de detecção de colisão utiliza intervalos de tempo discretos, isso significa que existe a possibilidade de algumas colisões não serem detectadas ou detectadas com atraso. Certamente, esta dificuldade complica o processo de resposta à colisão.

Diversas técnicas abordam o problema tentando diminuir o tempo gasto neste processo, entre as quais se destacam as técnicas que fornecem informação para o processo de resposta ou separação. As colisões são detectadas a partir da previsão de onde estarão os objetos no próximo tempo. Para isso, os objetos são movidos temporariamente para a posição onde estariam se não houvesse colisões. A partir desse momento, verifica-se se há interseção entre os objetos da cena.

A detecção de colisão é usualmente dividida em duas etapas. A primeira é uma detecção de colisão simplificada (*broad phase*), cujo objetivo não é a exatidão dos resultados, mas apenas a eliminação de pares de objetos que garantidamente não colidem. A segunda é a detecção de colisão exata (*narrow phase*), que permite detectar colisões reais, ou seja, o local exato onde há colisão. Esta etapa é a mais custosa já que é mais dependente da complexidade geométrica dos objetos envolvidos.

### 2.3.1 Volumes Limitantes

Os volumes limitantes são utilizados, principalmente, na fase de detecção de colisão simplificada para acelerar e facilitar a identificação das possíveis colisões entre objetos. Entende-se por volume limitante (*bounding volume*) uma forma simples como o menor volume capaz de envolver um objeto por completo. As formas geométricas mais comuns são esferas e caixas (paralelepípedos), que por sua vez podem ser de

dois tipos: alinhadas com os eixos, isto é com faces paralelas ou perpendiculares ao sistema de coordenadas globais, conhecidas por AABB (*Axis-Aligned Bounding Boxes*), ou de orientação arbitrária, isto é, com faces não alinhadas ao sistema de coordenadas globais, conhecidas OBB (*Oriented Bounding Boxes*). Estes elementos, além de servirem para os algoritmos de colisão, são importantes em outras operações.

### Esferas limitantes

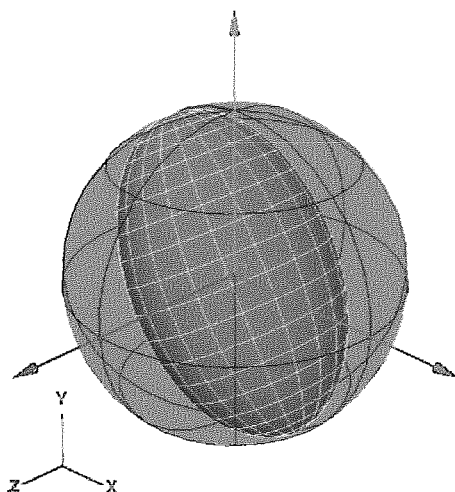


Figura 2.4: Esfera como volume limitante.

Esferas são provavelmente o modelo geométrico primitivo mais simples. A esfera pode ser representada por quatro valores escalares: três para designar o centro e um para o raio da esfera, podendo ser facilmente armazenada e ocupando pouca memória. Uma propriedade importante é o fato de serem invariantes a rotações o que as tornam boas candidatas a volume limitante de objetos rígidos. Além disso, devido à sua simplicidade, a detecção de colisão entre esferas é relativamente fácil de ser implementada.

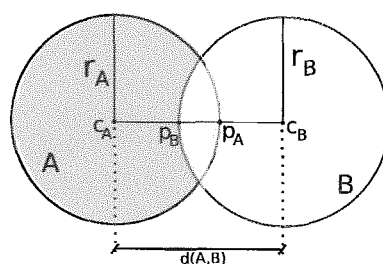


Figura 2.5: Duas esferas com interseção.

Considerando duas esferas  $A$  e  $B$ , existe uma colisão entre as duas se a distância

entre os dois centros  $c_A$  e  $c_B$ , respectivamente, é igual ou menor do que a soma dos raios das esferas  $r_A$  e  $r_B$ :

$$A \cap B \neq \emptyset \quad \equiv \quad |c_A - c_B| \leq r_A + r_B.$$

A profundidade de penetração entre as esferas é a soma dos raios menos a distância entre os centros. Se as esferas não se interceptam, a penetração é zero:

$$d(A, B) = \max(r_A + r_B - |c_A - c_B|, 0).$$

Para computar os pontos máximos de penetração de cada esfera  $p_A$  e  $p_B$ , considere que  $\vec{v} = c_A - c_B$ , onde  $\vec{v}$  é o vetor entre o centro de  $B$  e o centro de  $A$ . Então os pontos

$$p_A = c_A - r_A \frac{\vec{v}}{|\vec{v}|} \quad \text{e} \quad p_B = c_B + r_B \frac{\vec{v}}{|\vec{v}|}$$

são os pontos máximo de penetração de cada esfera. Note que estas expressões somente são válidas para esferas não concêntricas. Para um par de esferas concêntricas,  $\vec{v}$  é um vetor zero e então, a expressão resulta numa divisão por zero.

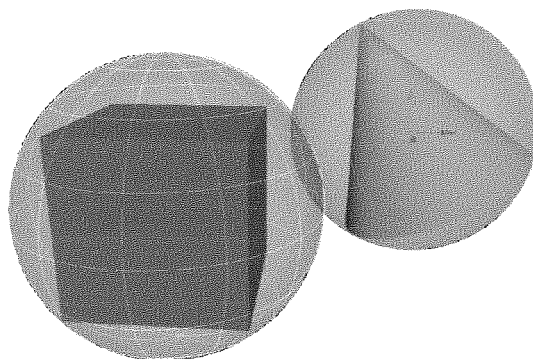


Figura 2.6: Duas esferas com interseção.

Contudo, para a maioria dos objetos, uma esfera não é considerada o melhor volume limitante, pois mesmo sendo a menor esfera possível, grande parte do volume da esfera não é ocupado pela forma do objeto. Assim, são identificados alguns pares de colisão que com outros volumes limitantes, mais justos aos objetos, não seriam considerados, filtrando ainda mais os pares de colisão para a próxima etapa.

### Caixas Limitantes de Orientação Arbitrária

As caixas limitantes de orientação arbitrária (OBBs) não são o tipo de volume limitante mais econômico em termos de custo de armazenamento e custo computacional

para verificação de interseção. Porém, este tipo de volume limitante, em contraste com os outros, aproxima de maneira bastante justa a superfície dos modelos, justificando o alto custo computacional e de armazenamento. A orientação da OBB é normalmente representada por uma matriz  $3 \times 3$ , a qual define uma orientação com respeito ao sistema de coordenadas do modelo limitado. Ainda que uma representação por ângulos de Euler ou por quatérnios possam ser usadas, é preferível usar uma matriz, já que esta é mais eficiente em verificações de interseção [12]. As informações da OBB são determinadas uma única vez para os corpos rígidos. Após determinada, a OBB será atualizada com a aplicação das mesmas rotações ou translações do corpo rígido.

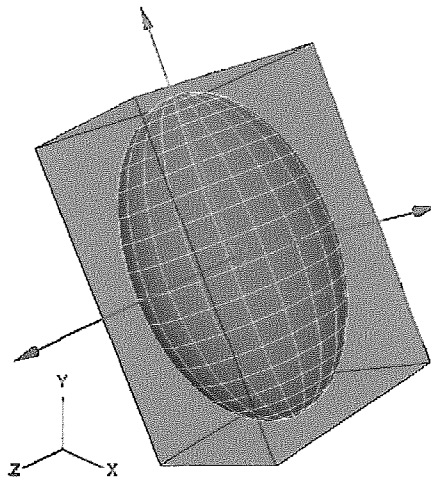


Figura 2.7: Caixas Limitantes de Orientação Arbitrária.

### Caixas Limitantes Alinhadas com os Eixos

As caixas limitantes alinhadas com os eixos (AABBs) são o tipo de volume limitante mais utilizado por ser facilmente determinado, ocupar pouco espaço de memória e suportar testes de interseção rápidos. AABBs são usadas ainda mais extensivamente do que esferas limitantes, embora precisem de mais espaço de armazenamento do que estas (na verdade, seis escalares). Com as AABBs, pode-se verificar uma interseção usando apenas seis operações primitivas. Porém, na média, elas também não costumam modelar formas muito justas [9].

Uma AABB é comumente representada por dois pontos extremos que consistem nos valores máximos e mínimos dos três eixos  $X, Y$  e  $Z$ . O teste de interseção,

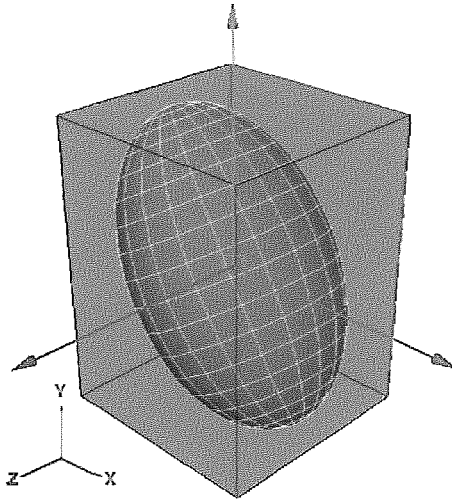


Figura 2.8: Caixa Limitante Alinhada com os Eixos.

portanto, corresponde à comparação entre os pontos extremos. Considerando duas AABBs  $A$  e  $B$ , existe interseção quando:

$$[A_{min}, A_{max}] \cap [B_{min}, B_{max}] \neq \emptyset \quad \equiv \quad A_{min} \leq B_{max} \text{ e } A_{max} \leq B_{min}.$$

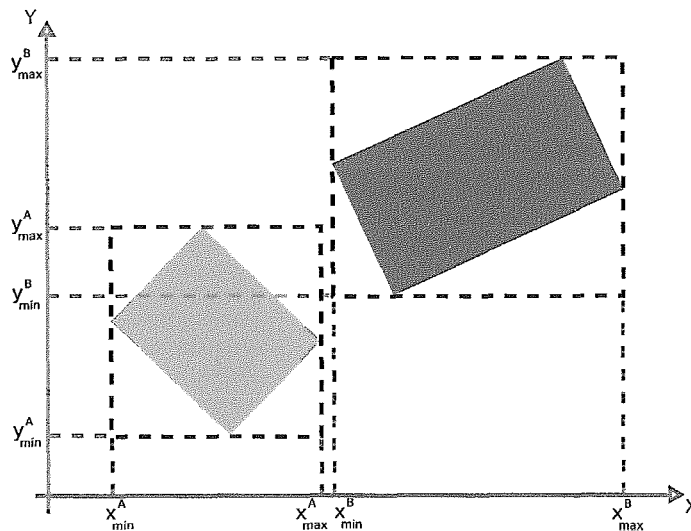


Figura 2.9: Duas AABBs com interseção em apenas um dos eixos.

Uma desvantagem das AABBs é a necessidade de verificar a estrutura geométrica do corpo rígido quando precisa ser atualizada. Dependendo da complexidade do corpo rígido, a atualização pode afetar diretamente o desempenho da detecção de colisão. Uma solução para gerar uma AABB mais rapidamente é utilizar as informações da OBB com as transformações já aplicadas. Contudo, a AABB assim computada não é tão justa quanto a gerada com base na geometria do corpo rígido.

## 2.3.2 Detecção de Colisão Simplificada

Apesar de verificações de interseção usando volumes limitantes serem relativamente baratas, verificar todos os  $\binom{n}{2}$  pares numa coleção de  $n$  volumes requer uma computação considerável. Numa situação comum, onde poucos objetos estão em colisão, um algoritmo sub-quadrático que consiga identificar os pares de objetos que se interceptam é preferível. Isto pode ser conseguido através da utilização de um particionamento espacial. A idéia, neste caso, consiste em restringir o teste de pares a objetos próximos que possam vir a colidir. Esta seção apresenta duas destas técnicas: a grade espacial e *sort and sweep* [13, 14], conhecida também por *sweep and prune*[15].

### Grade Espacial

Uma grade espacial ou simplesmente grade, é uma partição uniforme do espaço em células retangulares alinhadas com os eixos coordenados. Internamente, é um arranjo tridimensional de células de tamanho  $N_1 \times N_2 \times N_3$ , onde  $N_1$ ,  $N_2$ , e  $N_3$  são os números de subdivisões ao longo dos eixos coordenados. Cada célula mantém uma lista dos objetos que a interceptam. Portanto, inserir, mover ou remover um objeto na grade se resume em atualizar a lista dos objetos das células.

As grades são úteis para rejeitar rapidamente pares de objetos que não se interceptam. Porém, sofrem de um grande problema: objetos que ocupam muitas células prejudicam o desempenho da estrutura. Grades têm sido recomendadas para verificar as interseções em ambientes complexos [16] contendo milhares de objetos rígidos. Um algoritmo que usa grades deverá achar todos os pares de colisão de cada célula. Os benefícios de usar-se uma grade são: pouco uso de memória no armazenamento e acesso rápido às células.

A maior desvantagem das grades é que o desempenho é altamente dependente da densidade dos objetos no espaço, isto é, depende de como os objetos estão distribuídos no espaço. Por exemplo, se há muitos objetos agrupados em poucas regiões do espaço, então poucas células conterão a maioria dos objetos e a maior parte das células ficará vazia. Em tais casos, uma grade não é muito útil para a rejeição de pares de objetos na verificação de interseção.

Outro problema é escolher uma resolução adequada para a grade. Se usar uma

grade com poucas células, então as células ficarão muito cheias e muitas verificações de interseção serão executadas para pares de objetos distintos da célula. Se, por outro lado, a grade tiver uma resolução fina, então a grade precisará de muito espaço de armazenamento, e muitas células manterão referências a objetos idênticos.

Além disso, mover um objeto grande no espaço da grade pode ser bastante custoso, já que o objeto pode estar coberto por muitas células. Grades têm bom desempenho quando um modelo é composto por um conjunto de primitivas geometricamente coerentes com densidade e tamanho uniformes. Na prática, entretanto, poucos modelos satisfazem estas condições. Por conseguinte, esquemas de divisão adaptativa, usando divisão recursiva do espaço costumam ter melhor desempenho.

### ***Sort and Sweep***

É uma das técnicas mais utilizadas como detecção de colisão simplificada. Esta técnica baseia-se num algoritmo incremental que detecta um conjunto de volumes limitantes que se interceptam durante a simulação e, tem uma complexidade  $\Omega(n \log n)$  no pior caso. A idéia geral reside na observação de que se as projeções de dois objetos sobre uma linha reta não se interceptam, então pode-se garantir que os próprios objetos não se interceptam. Assim, o método consiste em calcular as projeções dos objetos da cena num certo número de retas, onde, normalmente, são empregadas projeções sobre os eixos coordenados. Para cada reta é mantida uma lista dos intervalos de projeção dos volumes limitantes (veja a Figura 2.10). Estas listas são atualizadas inserindo os intervalos de projeção para cada passo de tempo e ordenando-as pelo método de inserção. Normalmente, como os objetos movimentam-se pouco entre os quadros, cada lista mantém-se praticamente ordenada para o próximo quadro. Após a ordenação dessas listas, os intervalos de dois objetos são comparados e caso haja interseção nos três eixos, este par de objetos é marcado para um teste de colisão mais exato. Um aspecto importante do método é a escolha do volume limitante, sendo usualmente empregadas as AABBs e esferas limitantes.

### **2.3.3 Detecção de colisão exata**

Os métodos utilizados para a detecção de colisão exata podem ser classificados em cinco grupos: técnicas baseadas em hierarquias de volumes limitantes, técnicas de



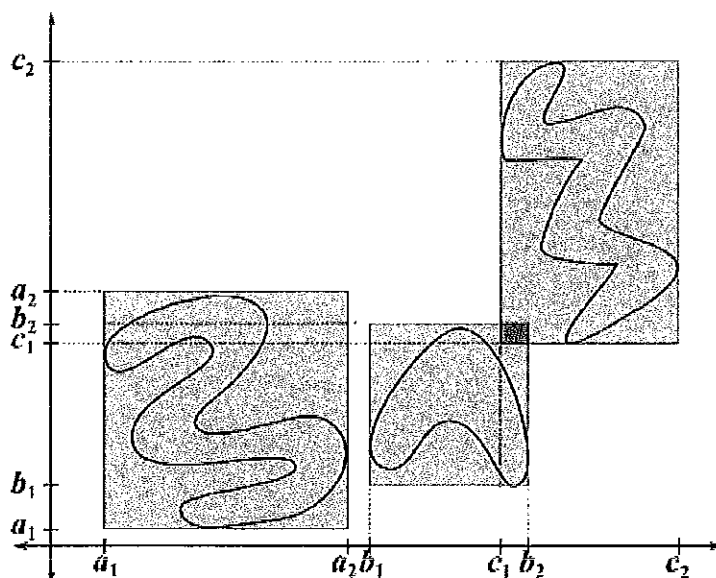


Figura 2.10: Método de detecção de colisão *sweep and prune* em 2D.

subdivisão espacial, técnicas que usam campos de distância, técnicas que usam o espaço da imagem e, recentemente, técnicas que se baseiam no uso de partículas.

### Hierarquias de Volumes Limitantes

Estruturas de dados hierárquicas baseadas em volumes limitantes têm alcançado excelentes resultados em animação de objetos rígidos, já que podem ser construídas numa etapa de pré-processamento. No entanto, estas estruturas não são adequadas para a simulação de objetos deformáveis, onde é necessária uma atualização a cada nova forma assumida pelo objeto durante a simulação.

A idéia básica nesta abordagem é dividir o objeto recursivamente obtendo como resultado uma árvore. Os nós-folha referem-se às primitivas do objeto. Cada nó da árvore possui um volume limitante que garantidamente contenha todas as primitivas contidas nos nós-filho descendentes. Este volume limitante é utilizado para fazer verificações rápidas de interseção como condição para descer na hierarquia.

Existem vários tipos de hierarquias de volumes limitantes, e no processo de detecção de colisão, as hierarquias usualmente são verificadas de cima para baixo testando, recursivamente, a interferência entre os volumes limitantes contidos em cada par de nós. Se ambos os nós são folhas, é feito um teste exato entre primitivas. Se um nó é folha e o outro é interno, o volume limitante do nó folha é testado contra o dos filhos do nó interno. Para aperfeiçoar este processo, se ambos os nós

são internos, é recomendado verificar o nó com menor volume limitante versus os filhos do nó maior.

As hierarquias de volumes limitantes mais comuns utilizam caixas limitantes alinhadas com os eixos (ver Figura 2.11), esferas limitantes (ver Figura 2.12) e caixas limitantes de orientação arbitrária (ver Figura 2.13).

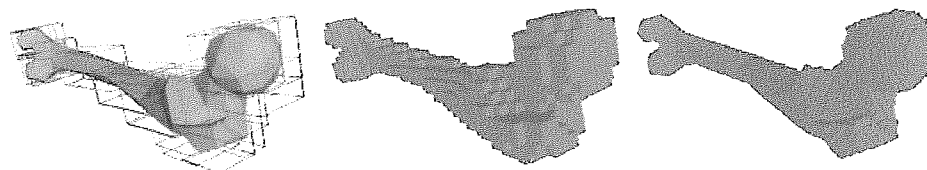


Figura 2.11: Hierarquia de caixas limitantes alinhadas com os eixos (*AABB-Tree*).

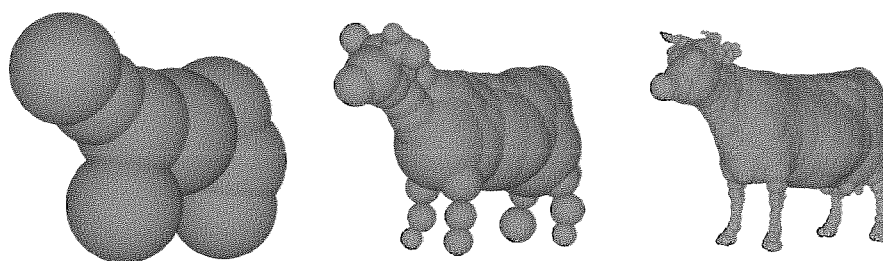


Figura 2.12: Hierarquia de esferas limitantes (*Sphere-Tree*).

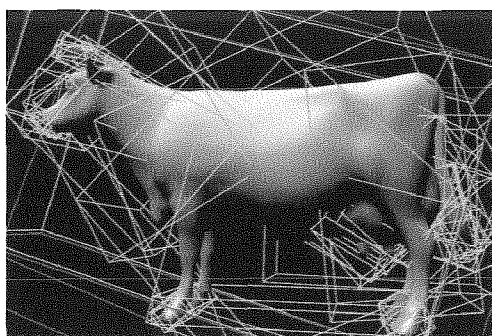


Figura 2.13: Hierarquia de caixas limitantes orientadas (*OBB-Tree*).

## Subdivisão Espacial

Em contraste com as técnicas de volumes limitantes que dividem o espaço do objeto, as técnicas de subdivisão espacial dividem em células o espaço onde os objetos

estão imersos, ou seja, o cenário onde os objetos interagem. Cada célula da partição mantém uma lista das primitivas que podem ser completa ou parcialmente contidas na célula. A principal consideração consiste na escolha de uma estrutura de dados flexível e eficiente na atualização e no uso de memória. Diversos esquemas de subdivisão espacial têm sido propostos, entre eles, as grades uniformes [17, 18], *octrees* [19], *BSP-trees* [20], *kd-trees* [21] e tabelas de dispersão espacial [22, 23].

Na construção destas estruturas de dados, a distribuição espacial dos objetos pode ou não ser considerada na geometria da subdivisão. Entre as primeiras incluem-se as *BSP-Trees* e *kd-trees*, por exemplo, que dividem o espaço 3D baseado na orientação e na posição de algumas primitivas escolhidas. Já em abordagens que não dependem dos objetos, como *octrees* e grades regulares, por exemplo, a subdivisão sempre ocorre segundo planos pré-determinados. Na verdade, a diferença reside na geometria de partição.

A detecção de colisão baseada em subdivisão espacial requer que as primitivas de todos os objetos sejam discretizadas em células. Assim, para cada célula que contém mais de uma primitiva, faz-se uma verificação de interseção a fim de encontrar colisões entre as primitivas. Quando objetos deformáveis são considerados, é necessário atualizar a estrutura de dados a cada passo de tempo. Teschner et al. [22] apresentaram uma idéia simples para evitar atualizações desnecessárias. A idéia é manter em cada célula uma variável indicando o último instante de tempo (*timestamp*) em que esta foi atualizada. Mais tarde, as verificações de interseção são feitas apenas nas células que foram atualizadas no passo de tempo corrente.

## Técnicas no Espaço da Imagem

Inicialmente estas técnicas foram usadas apenas para detectar interferência. Aproveitando a funcionalidade conhecida como *Occlusion Query* que foi incorporada às primeiras unidades de processamento gráfico programáveis – mais conhecidas como GPUs (*Graphics Processing Units*). Esta funcionalidade realiza consultas de visibilidade, reportando o número de pixels visíveis após uma exibição. Com a evolução das GPUs, a verificação de interferência passou a ser feita inspecionando diretamente os *buffers* das GPUs.

Estas técnicas não precisam de estruturas de dados complexas já que operam

em imagens obtidas após o processo de exibição da geometria. Entretanto, como a verificação de interferência é feita na resolução do espaço da imagem, os resultados são aproximados e suscetíveis a erros de precisão. Um trabalho pioneiro foi apresentado por Shinya et al. [24], onde é descrita uma técnica para detecção de colisão entre objetos rígidos. Partes da superfície dos objetos eram geradas em buffers de profundidade e guardadas em imagens, que posteriormente eram empregadas para executar testes de interseção através de operações booleanas. Esta técnica é simples e rápida, porém suporta apenas objetos convexos.

Heidelberger et al. [25] apresentaram uma extensão da técnica de Shinya et al. capaz de tratar objetos deformáveis. De forma similar, esta técnica usa imagens de profundidade (*Layered Depth Images*), onde as imagens são usadas para armazenar entradas e saídas de raios paralelos lançados a partir de diversos pontos de vista. Esta técnica trata objetos côncavos e apresenta desempenho em tempo real para objetos simples.

No entanto, as técnicas baseadas no espaço da imagem têm algumas desvantagens. Por exemplo, a necessidade de processar na CPU as imagens obtidas da GPU esbarram na limitação de transferência de dados entre CPU-GPU-CPU que depende das dimensões do espaço da imagem suportado pela GPU. Para superar o problema de transferência de dados, Govindaraju et al. apresentaram uma técnica chamada CULLIDE [26], que permite filtrar pares de objetos em colisão potencial usando consultas de visibilidade (*occlusion queries*). Esta técnica, entretanto, requer uma pré-computação ou etapa de configuração não trivial. Posteriormente, Govindaraju et al. apresentaram uma abordagem de pré-computação mais eficiente para o CULLIDE, chamada decomposição cromática [27]. Esta técnica é limitada para malhas poligonais com conectividade fixa e depende do uso de várias direções de visualização para garantir uma detecção de colisão sem erros.

Além desses problemas, o custo de exibição pode prejudicar o desempenho dessas técnicas, já que usualmente é necessário exibir cada par de objetos em colisão potencial. Knott and Pai [28] propuseram um algoritmo que trata vários objetos numa exibição só, porém, a exibição é em modo aramado (*wireframe*), o que não garante uma detecção de colisões eficiente.

Um apanhado das técnicas que usam o espaço da imagem foi apresentado por

Teschner et al. [29], mostrando que estas técnicas são adequadas para serem usadas em ambientes dinâmicos com objetos deformáveis, já que, em geral, não requerem de pré-processamentos ou estruturas de dados complexas tais como hierarquias de volumes limitantes.

Recentemente, Hang-Yong et al. [30, 31] apresentaram técnicas para tratar múltiplos objetos numa exibição só, conseguindo ao mesmo tempo detectar pontos de contato, que podem ser aproveitados no processo de resposta às colisões.

### Detecção de Colisão Baseada em Partículas

Em contraste com as técnicas que usam o espaço da imagem, Harada [32] apresentou uma técnica de detecção de colisões baseada em partículas. A técnica se baseia na representação de objetos por partículas e no gerenciamento de uma grade espacial que mapeia estas partículas.

Os objetos são decompostos em partículas (pequenas esferas de igual tamanho). Para gerar uma amostra de partículas do modelo, é necessário um processo que basicamente discretiza o espaço do objeto numa grade onde uma partícula é atribuída a cada célula, sendo escolhidas apenas partículas que estão dentro do modelo. Assumindo que o modelo é uma malha fechada, pode-se usar a técnica de traçado de raios (*Ray-Casting*) o que permite obter pontos onde o raio intercepta o modelo. Com esta técnica, são detectados os pontos de entrada e de saída na malha. A figura 2.14 ilustra o processo. A resolução do conjunto de partículas depende do tipo de aplicação, já que uma representação muito densa do objeto pode diminuir o desempenho da simulação, e uma representação muito grossa, pode não aproximar corretamente o objeto.

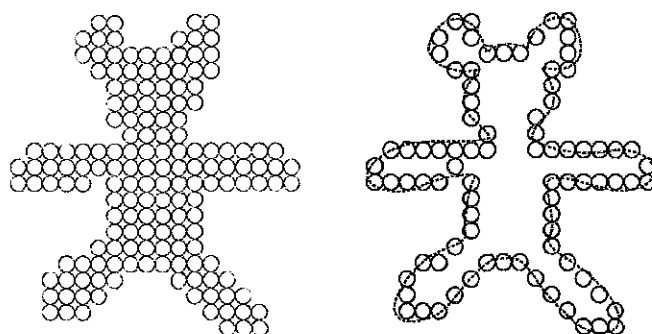


Figura 2.14: Partículas representando a geometria de um objeto.

A grade gerada é um array 3D onde cada célula, contém uma lista de índices de partículas. A célula é representada por um cubo, com largura e altura iguais ao diâmetro da partícula. A grade é atualizada a cada instante de tempo, mas apenas nas células onde houve movimento de partículas.

Cada partícula é mapeada na célula segundo a sua posição. O mapeamento é simples usando a função  $(g_x, g_y, g_z) = (\lfloor \frac{p_x}{L} \rfloor, \lfloor \frac{p_y}{L} \rfloor, \lfloor \frac{p_z}{L} \rfloor)$ , onde  $g_x$ ,  $g_y$  e  $g_z$  são as coordenadas na grade 3D;  $p_x$ ,  $p_y$  e  $p_z$  são as coordenadas da posição da partícula; e  $L$  é o diâmetro da partícula.

A detecção de colisão resume-se à verificação de interseção entre esferas, ou seja, há colisão se a distância entre os centros de duas partículas é menor que seu diâmetro. Este processo é auxiliado pela grade 3D que restringe o teste a partículas que residem numa mesma célula ou nas células adjacentes. Assim, a execução deste processo tem complexidade  $O(n)$ .

## 2.4 Simulação de Corpos Rígidos

O estudo da simulação física envolvendo objetos rígidos foi foco de diversas pesquisas por um longo tempo, entre elas as contribuições de Baraff et al. [33, 4], Popovic et al. [34], Guendelman et al. [10], entre outras.

O método descrito nesta seção baseia-se na técnica apresentada por Guendelman et al. [10] e que tornou-se bastante popular, sendo usada em diversas bibliotecas de física para jogos. Esta técnica permite uma simulação de objetos rígidos com resultados visualmente plausíveis, suportando múltiplos contatos e configurações com objetos empilhados.

O sucesso da técnica pode ser atribuído à combinação adequada das partes que a compõem: representação geométrica, detecção de interferência, integração, detecção de colisões, cálculo de contatos, grafo de contato, propagação de choque e a separação.

### 2.4.1 Representação do Objeto

A representação do objeto compreende a representação da superfície e suas propriedades, tipicamente massa, centro de massa, densidade e tensor de inércia. A

superfície do objeto usualmente é representada por uma malha triangulada e as propriedades são pré-computadas a partir dela. Para facilitar os cálculos, o espaço do objeto é configurado para que o centro de massa coincida com a origem e para que os eixos  $X, Y$  e  $Z$  estejam alinhados com os eixos principais do tensor de inércia. Observe que nesta configuração o tensor de inércia deve ser uma matriz diagonal, o que permite simplificar cálculos durante a fase de integração.

Com o objetivo de acelerar o processo de detecção de colisão, para cada objeto é computada uma função de distância que permite fazer consultas rápidas de colisão e interferência. Esta função pode ser representada por uma *grade* ou uma *octree*.

## 2.4.2 Detecção de Interferência

O processo de detecção de colisão, como descrito na seção 2.3, compreende duas etapas: a primeira visa detectar pares de objetos em colisão potencial e a segunda que executa verificações de colisão exatas. Por simplicidade, para a etapa de filtragem grosseira, escolheu-se usar esferas envolventes permitindo verificação  $O(N^2)$  entre todos os objetos. Naturalmente, esta etapa pode ser melhorada usando-se métodos mais eficientes.

Já a etapa de detecção de colisão exata visa obter uma lista de pontos de colisão. Para tanto, é usada a função de distância associada a cada objeto, que permite obter os vetores normais à superfície nos pontos de contato (ver a figura 2.15). As normais e os pontos de contato são usados no cálculo de impulsos.

Para cada par de objetos  $A$  e  $B$  em colisão potencial, os vértices  $v_A(v_B)$  do objeto  $A(B)$  são avaliados com a função de distância do objeto  $B(A)$ . Se este retorna sinal negativo, então o vértice  $v_A(v_B)$  está dentro do objeto  $B(A)$ , e portanto o vértice é inserido na lista de pontos em colisão. Esta técnica não é suficiente para detectar todas as colisões. Por exemplo, arestas compridas que possuem vértices muito distantes podem eventualmente se interpenetrar (veja a figura 2.16). Para tratar este problema, nas arestas são armazenados pontos extras, que também são verificados para interferência.

Outro problema inerente da integração está relacionado ao passo de tempo ( $\Delta t$ ) empregado. Por exemplo, se  $\Delta t$  é grande, objetos pequenos podem se atravessar. Este problema é tratado limitando o passo de tempo baseado nas velocidades

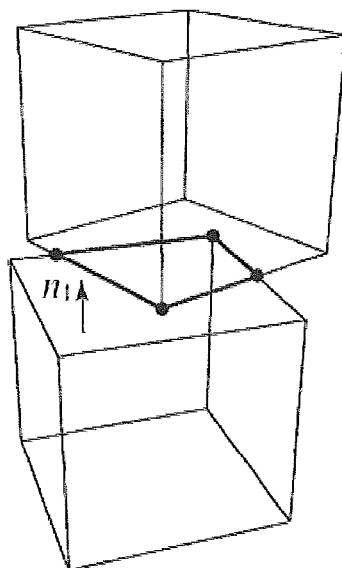


Figura 2.15: contatos entre objetos empilhados.

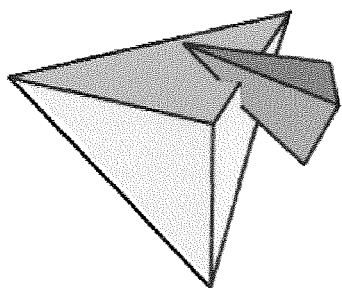


Figura 2.16: interferência entre objetos com arestas não proporcionais.

(translacional e rotacional) e no tamanho dos volumes envolventes dos objetos sendo simulados.

### 2.4.3 Integração

No processo de integração, para cada objeto são usadas as equações:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \quad \frac{d\mathbf{q}}{dt} = \frac{1}{2}\omega\mathbf{q}, \quad (2.26)$$

$$\frac{d\mathbf{v}}{dt} = \frac{\mathbf{f}}{m}, \quad \frac{d\mathbf{L}}{dt} = \boldsymbol{\tau}, \quad (2.27)$$

onde  $\mathbf{x}$  e  $\mathbf{q}$  são sua posição e orientação (representada por um quatérnio),  $\mathbf{v}$  e  $\omega$  são suas velocidades linear e angular,  $\mathbf{f}$  e  $\boldsymbol{\tau}$  são a força e o torque aplicadas sobre ele,  $m$  é sua massa e  $\mathbf{L} = \mathbf{I}\omega$  é seu momento angular que depende de seu tensor de inércia.



O tensor de inércia para cada passo de tempo é computado via  $\mathbf{I} = \mathbf{R}\mathbf{I}_{body}\mathbf{R}^T$ , onde  $\mathbf{R}$  é a matriz de orientação e  $\mathbf{I}_{body}$  é o tensor de inércia inicial. Como mencionado anteriormente,  $\mathbf{I}_{body}$  é uma matriz diagonal, o que simplifica os cálculos. Existem diversos métodos para integração e seu uso depende do tipo de aplicação. Em particular, para uma simulação com muitos objetos e múltiplos contatos é preferível usar o método de integração “para frente” (*forward integration method*) de Euler:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \frac{1}{2}\Delta t\omega_n\mathbf{q}_n, \quad (2.28)$$

aplicando-o na equação 2.26 de orientação.

A idéia desta abordagem é computar o passo de tempo seguinte e, subseqüentemente, tratar as colisões e os contatos. De modo geral, as colisões produzem impulsos, que descontinuamente modificam a velocidade, e os contatos são associados a forças e acelerações. Assim, o algoritmo de integração é combinado com o processamento de colisões e reúne os seguintes passos:

- detecção de colisão;
- atualização das velocidades usando as equações de velocidade e torque;
- resolução de contatos;
- atualização das posições usando as equações de posição e orientação.

#### 2.4.4 Tratamento de Colisões

Quando há muitos objetos em movimento, o tratamento eficiente de colisão é complexo e custoso, especialmente ao se considerar a ordem cronológica. O método apresentado por Guendelman et al. [10] resolve simultaneamente todas as colisões. Embora esta técnica não forneça os mesmos resultados que uma resolução de colisões em ordem cronológica, ela conduz a uma solução fisicamente plausível.

As colisões são detectadas prevendo onde os objetos estarão no passo de tempo seguinte. Assim, os objetos são movidos temporariamente no passo de tempo seguinte e verifica-se a interferência. A mesma técnica é usada para detectar contatos e, se não há colisão, é desejável que os objetos sejam deslocados para a mesma

posição em ambas as etapas. Para garantir este resultado, são usadas as novas velocidades para prever as posições dos objetos em ambas as etapas. Por outro lado, enquanto as velocidades antigas são usadas para resolver as colisões, as velocidades novas são usadas para resolver os contatos. Por exemplo, para a fase de colisão, se a posição e a velocidade de um objeto são  $\mathbf{x}$  e  $\mathbf{v}$ , a verificação de interferência é feita usando a posição predita  $\mathbf{x}' = \mathbf{x} + \Delta t(\mathbf{v} + \Delta t\mathbf{g})$  e são aplicados impulsos atualizando a velocidade corrente. Durante o processamento de contatos, é usada a posição predita  $\mathbf{x}' = \mathbf{x} + \Delta t\mathbf{v}'$  e impulsos são aplicados a esta nova velocidade  $\mathbf{v}'$ .

A estrutura do algoritmo consiste em mover todos os objetos rígidos para suas posições preditas e só então identificar e processar pares de objetos em colisão potencial. Como as velocidades dos objetos mudaram devido às colisões, entretanto, podem ocorrer novas colisões entre pares de objetos que não existiam anteriormente. Por conseguinte, o processo todo é repetido um número pequeno de vezes.

Para obter um resultado mais natural, a cada iteração os pares de objetos em colisão potencial são coletados numa lista e embaralhados de forma a evitar um processamento similar à iteração anterior.

Num ponto de colisão, a velocidade local de um objeto é  $\mathbf{u} = \mathbf{v} + \boldsymbol{\omega} \times \mathbf{r}$ , onde  $\mathbf{r}$  é o vetor que representa a **posição relativa** do ponto de colisão em relação ao centro de massa. Aplicando um impulso neste ponto de colisão, a velocidade linear e angular muda de acordo com

$$\mathbf{v}' = \mathbf{v} + \frac{\mathbf{j}}{m} \quad (2.29)$$

e

$$\boldsymbol{\omega}' = \boldsymbol{\omega} + \mathbf{I}^{-1}(\mathbf{r} \times \mathbf{j}). \quad (2.30)$$

Assim, a nova velocidade no ponto de contato é

$$\mathbf{u}' = \mathbf{u} + K\mathbf{j}, \quad (2.31)$$

onde  $K = \frac{1}{m} + \mathbf{r}^{*T}\mathbf{I}^{-1}\mathbf{r}^*$ ,  $\mathbf{1}$  é a matriz de identidade, e “\*” indica a matriz anti-

simétrica<sup>1</sup> associada ao vetor.

Seja  $\mathbf{u}_{rel} = \mathbf{u}_1 - \mathbf{u}_2$  a velocidade relativa entre os objetos  $O_1$  e  $O_2$  no ponto de colisão. Um impulso  $\mathbf{j}$  é aplicado a cada objeto com sinal oposto, isto é,  $+\mathbf{j}$  no objeto  $O_1$  e  $-\mathbf{j}$  no objeto  $O_2$ .

## 2.4.5 Tratamento de Contatos

Após ter processado todas as colisões, os objetos apresentam um comportamento plausível, porém colisões ainda podem existir. Neste caso é executado o processo de resolução de contatos. O objetivo deste processo é resolver as forças entre os objetos. Primeiro, as velocidades dos objetos são atualizadas e, como no processo de resolução de colisões, os contatos são detectados predizendo a posição dos objetos no passo de tempo seguinte, ignorando as forças de contato.

A figura 2.17 ilustra uma cena com objetos empilhados mostrando o processo de resolução de contatos. Após iniciada a simulação, pode-se ver que apenas a caixa de baixo terá interferência. Entretanto, após processar as forças nesta caixa, ela colidirá com a caixa de cima e, assim sucessivamente até a última caixa no topo da pilha. Em geral, este processo de propagação separa todos os objetos após algumas iterações. Naturalmente, em algumas situações o processo pode não convergir, como no caso de objetos empilhados. Este problema é tratado usando um grafo de contato.

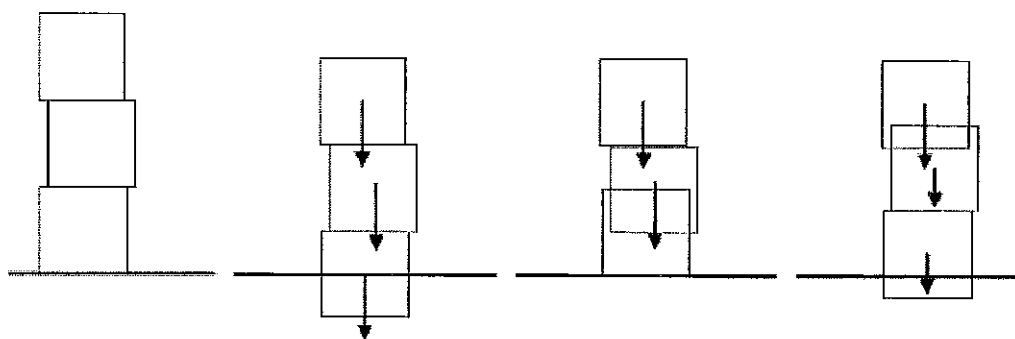


Figura 2.17: exemplo de objetos empilhados.

---

<sup>1</sup>Um vetor  $\mathbf{u}$  possui uma matriz associada  $\mathbf{u}^* = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}$ , de forma que  $\mathbf{u}^* \mathbf{v} = \mathbf{u} \times \mathbf{v}$ .

### 2.4.6 Grafo de Contato

Após a atualização de velocidades dos objetos é construído um grafo com o propósito de identificar objetos ou grupos de objetos que estão acima de outros objetos e determinar uma ordem apropriada. Os seguintes passos são necessários:

- construir o grafo. Para cada objeto  $i$ , computa-se uma nova posição a partir da equação  $\mathbf{v}' = \mathbf{v} + \Delta t \mathbf{g}$ , sendo as posições dos demais objetos ( $j \neq i$ ) computadas usando-se as velocidades antigas. Adicionalmente, para cada objeto  $j$  interceptado por  $i$ , criar uma aresta direcionada  $j \leftarrow i$  indicando que  $i$  está em cima de  $j$ ;
- eliminar ciclos. O conjunto de pares no grafo deve ser único;
- atribuir uma ordem. Usando um método de ordenação topológica, atribuir a cada par de objetos um nível para o processamento de contato.

As figuras 2.18 (a) e (b), mostram exemplos de grafos para duas cenas.

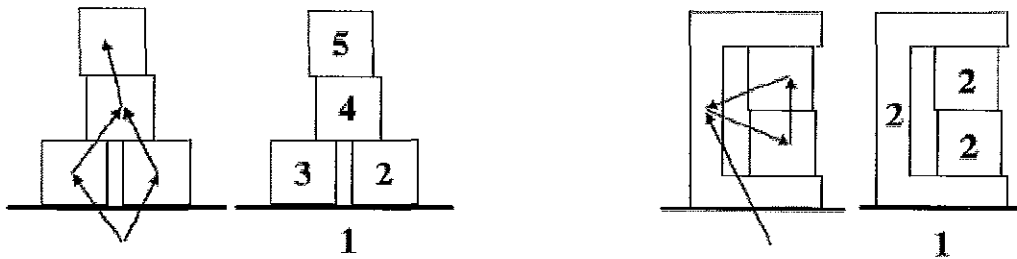


Figura 2.18: grafo de contato para cenas com objetos empilhados.

### 2.4.7 Propagação de Choque

Mesmo com a ajuda do grafo de contato, o modelo de propagação para os contatos requer muitas iterações para produzir um resultado visualmente realista, especialmente em situações onde há muitos objetos empilhados. A técnica de propagação de choque (figura 2.19) inicia com os objetos de baixo, ou seja, objetos de menor nível no grafo de contatos. Logo, objetos processados terão massa infinita e velocidade zero evitando que objetos de cima empurrem objetos de baixo.

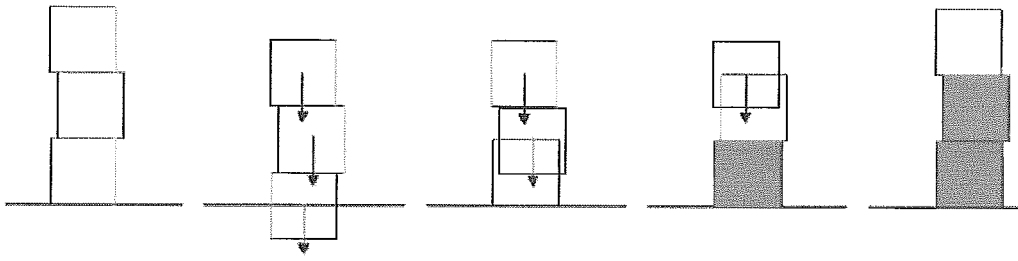


Figura 2.19: contatos entre objetos empilhados.

### 2.4.8 Separação dos Objetos

O processo de separação segue com o cálculo das posições após terem sido feitos o tratamento de colisões e o tratamento de contatos usando o grafo de contatos e a propagação de choque. Para reduzir a margem de erro, a cada nível do grafo os objetos são separados gradualmente incrementando a fração de interpenetração. Em alguns casos ainda podem existir colisões por erros de arredondamento e dependências cíclicas no grafo.

Em resumo a seqüência de passos durante a simulação é:

1. detectar interferência e aplicar impulsos de colisões;
2. computar um grafo de contato;
3. integrar as velocidades usando forças externas;
4. detectar interferência e aplicar impulsos de contato inelástico;
5. aplicar propagação de choque;
6. integrar as posições.

## Capítulo 3

# Desacoplamento da Simulação

Como explicado anteriormente, a animação física consiste em uma animação gerada a partir de quadros produzidos por uma simulação das leis da física. Esses quadros são imagens estáticas que, se exibidos em determinada velocidade, passam a sensação de movimento da cena.

Essa sensação de movimento ocorre por causa da persistência da visão [35]. O olho humano assimila a sequência de quadros e interpreta-os como um movimento contínuo. A persistência do movimento é criada com a apresentação de uma sequência de imagens em uma velocidade suficiente para induzir a sensação de um movimento contínuo. A taxa capaz de produzir esta percepção é, usualmente, de vinte e quatro quadros por segundo.

Na verdade, existem duas taxas importantes na animação: uma corresponde ao número de quadros por segundo exibidas, já a outra corresponde ao número de quadros diferentes por segundo. Por exemplo, uma animação pode exibir trinta quadros por segundo, mas somente cinco são diferentes, ou seja, cada um deles é exibido seis vezes. O resultado é uma animação sem a sensação de um movimento natural e suave, diferente de uma com trinta quadros diferentes. Portanto, quanto maior o número de quadros diferentes, melhor é a sensação de movimento.

Normalmente, o modelo tradicional implementado para a animação física apresenta uma taxa baixa de quadros diferentes por segundo, a qual resulta na perda da sensação de movimento dos objetos da cena. A idéia principal do modelo proposto é tentar minimizar esta perda de sensação de movimento.

Neste capítulo, são descritos: o Modelo Tradicional, para uma compreensão do

cenário usual, e o Modelo Proposto, com as alterações necessárias para atender à proposta.

### 3.1 Modelo tradicional

O modelo de processamento tradicionalmente implementado para a animação física é o seqüencial, ou seja, a exibição de um quadro deste tipo somente é feita após o término do cálculo dos parâmetros físicos dos objetos. Para isso, inicia-se um ciclo constituído de duas partes: o cálculo do quadro e a exibição do quadro. O cálculo do quadro é responsável por determinar a situação dos objetos considerando a atuação das leis da física em um intervalo de tempo  $\Delta t$ . A etapa de exibição corresponde à exibição propriamente dita dos quadros.

Nesse modelo,  $\Delta t$  corresponde ao tempo decorrido para concluir o ciclo anterior. Portanto,  $\Delta t$  para o cálculo do quadro seguinte  $n$  é determinado por:

$$\Delta t^n \approx T_C^{n-1} + T_E^{n-1}$$

em que  $T_C$  é o tempo decorrido para calcular o quadro e  $T_E$  é o tempo decorrido para exibir o quadro. Um exemplo pode ser observado na figura 3.1.

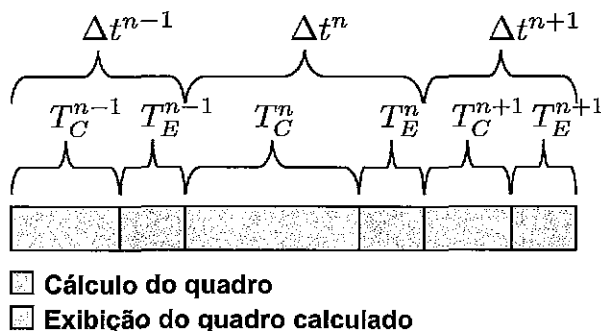


Figura 3.1: Exemplo da definição do  $\Delta t$

$T_E$  está diretamente relacionado à complexidade geométrica dos objetos a serem exibidos e independe da posição dos objetos ou da configuração da cena. Como os objetos não sofrem deformações geométricas entre os quadros por tratar-se de corpos rígidos,  $T_E$  pode ser considerado aproximadamente constante. Já  $T_C$  está diretamente relacionado à complexidade da cena e do número de objetos. Como a configuração da cena varia de um ciclo para o outro,  $T_C$  varia entre cada quadro.

Por exemplo, se uma cena tiver muitos objetos e todos próximos uns dos outros,  $T_C$  para este caso provavelmente será maior do que para o quadro seguinte, quando os objetos estiverem mais espalhados na cena. Portanto, pode-se considerar que  $\Delta t$  é diretamente proporcional a  $T_C$ , como pode ser observado na Figura 3.1.

O ideal para uma simulação é mostrar o resultado em tempo real. Deste modo, o resultado final do cálculo do quadro deve representar exatamente a situação da cena para o tempo decorrido no momento observado. Já para uma animação, o ideal é exibir os quadros em uma taxa suficiente para que os movimentos sejam suaves e naturais ao olho humano. Conforme mencionado anteriormente, esta sensação é alcançada com uma taxa igual ou superior a vinte e quatro quadros por segundo.

Conseqüentemente, a situação ideal para uma animação física é atender aos dois critérios citados. Neste caso, são necessários vinte e quatro ciclos por segundo, o que consiste em calcular um quadro e exibi-lo em 0,042 segundos, aproximadamente. Considerando-se  $\Delta t$  constante, esta situação será atendida quando  $T_C + T_E \leq \Delta t$ . Contudo, o ciclo seguinte não começa com fim do ciclo anterior, o que possibilita haver um tempo de ociosidade entre um quadro e outro como pode ser observado na Figura 3.2.

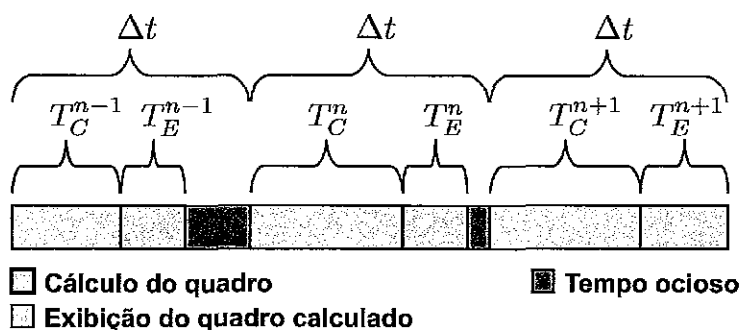


Figura 3.2: Exemplo de uma situação ideal

Entretanto, ainda não é possível ter um resultado em tempo real para configurações mais complexas da cena, devido à computação requerida. Assim, dependendo das alterações no cenário,  $T_C$  pode variar muito entre ciclos o que corresponde a uma variação de  $\Delta t$  para cada quadro calculado, de acordo com:

$$T_C + T_E \leq 0,042s \Rightarrow \Delta t = 0,042s$$

$$T_C + T_E > 0,042s \Rightarrow \Delta t = T_C + T_E$$



A Figura 3.3 mostra um exemplo dessa situação. Considera-se que a linha pontilhada representa o momento ideal para exibir o quadro de modo a manter a continuidade do movimento. Assim, os quadros  $n$  e  $n + 1$  tiveram o ciclo encerrado antes do momento ideal de exibição. Porém, o ciclo do quadro  $n - 1$  encerrou com atraso, o que representa uma perda momentânea do movimento dos objetos na cena e impossibilitou que o usuário interagisse com os objetos. Apesar disso, o quadro  $n$  representa a situação para o momento de exibição porque o tempo total decorrido até então foi considerado no cálculo.

Contudo, quanto maior for  $\Delta t$  utilizado, maior é a possibilidade do resultado do cálculo apresentar uma imprecisão. Isto pode ocorrer durante a tentativa de separar os objetos no tratamento de colisões devido à profundidade de penetração entre os objetos. Assim, considerando a figura 3.3, o quadro  $n$  pode apresentar imprecisões no resultado por causa de  $\Delta t^{n-1}$ .

A imprecisão não é o único problema identificado devido ao intervalo de tempo, pois, quanto menor for  $\Delta t$ , maior é o processamento necessário para calcular um período de tempo com alterações significativas na cena, o que caracteriza um desperdício de processamento. Além disso,  $T_C$  é normalmente menor nessas situações. Com isto, existe a possibilidade de apresentar maiores intervalos de tempo ociosos, impossibilitando a melhor utilização dos recursos disponíveis. Estas situações são representadas na figura 3.3 para  $\Delta t^n$  ou  $\Delta t^{n+1}$ .

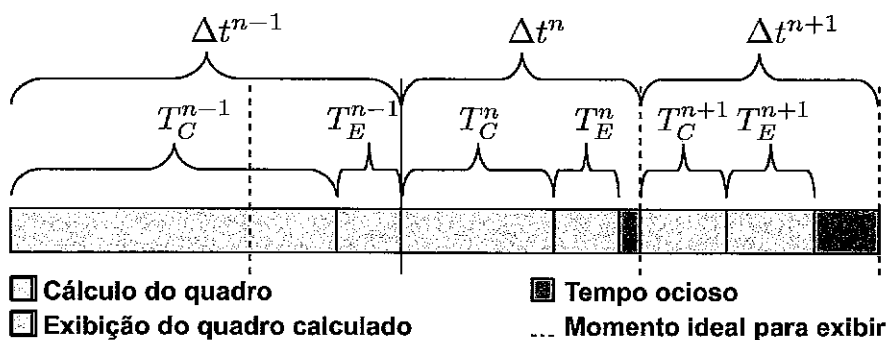


Figura 3.3: Exemplo de uma situação geral

Uma das soluções possíveis para a imprecisão é dividir  $\Delta t$  em períodos de tempo menores, processá-los, sequencialmente, um a um e obter o resultado para o período total. Porém, esta solução não é a ideal, pois aumenta o processamento causando o problema de desperdício mencionado anteriormente. A solução ideal é ter um

controle maior sobre o intervalo de tempo utilizado no cálculo para poder ajustá-lo de acordo com a necessidade.

## 3.2 Modelo Proposto

O modelo para a animação física proposto neste trabalho permite calcular os parâmetros físicos desacoplados da exibição em si. A idéia é permitir o cálculo de cada quadro da animação com passos de tempo  $\Delta t$  variados e não determinados pela exibição na tela. Como pode ser observado na figura 3.4, o processo de exibição e o de cálculo são executados ao mesmo tempo.

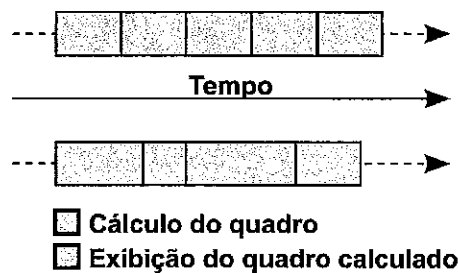


Figura 3.4: Demonstração da execução dos processos em paralelo

Com os dois processos em paralelo, podem aproveitar-se melhor os recursos para calcular quadros a serem exibidos em instantes de tempo futuros e, então, eliminar os tempos de ociosidade característicos do modelo tradicional. Assim, é possível já ter um quadro pronto para o momento de exibição, o que evita a perda da continuidade no movimento dos objetos por não ser preciso esperar o fim do cálculo do quadro. Isso também possibilita que seja utilizado mais tempo para calcular quadros mais complexos sem atrasar a animação em relação ao tempo real.

Além disso, este modelo permite uma maior flexibilidade para  $\Delta t$  o que não é possível no modelo tradicional por ser muito dependente do intervalo de tempo do ciclo anterior. Esta vantagem possibilita solucionar os problemas de imprecisão e desperdício. Contudo, o ideal é definir um valor que possibilite calcular o máximo de quadros para instantes futuros com o mínimo de processamento necessário e apresentando resultados plausíveis, ou seja, um valor que não cause muita imprecisão e reduza o desperdício de recursos. Determinar este valor é complicado, pois di-

ficilmente existirá um valor ideal para atender todas as configurações possíveis da cena.

Portanto, opta-se, inicialmente, por solucionar o problema do desperdício para melhorar a sensação de movimento da cena. Para tanto, atribui-se a  $\Delta t$  um valor que permita o cálculo de quadros para instantes futuros, evitando a imprecisão.

Como mencionado no capítulo anterior, as representações dos objetos incluem diversas propriedades físicas para o cálculo das leis da física. A partir deste cálculo, é determinado um estado do objeto que será exibido na animação. Portanto, considera-se como um estado do objeto o conjunto destas propriedades físicas e das propriedades de exibição para um dado momento. As propriedades de exibição consistem, normalmente, na cor ou textura do objeto, bem como sua geometria (vértices e faces, por exemplo).

Com esta estrutura de implementação, é necessário considerar dois tempos distintos: o tempo atual da simulação física  $T_F$  e o tempo atual de exibição  $T$ . Isto é,  $T_F$  corresponde ao somatório de todos os  $\Delta t$  utilizados no cálculo das leis físicas.

Assim, quando  $T_F$  for maior que  $T$ , caracteriza-se a existência de quadros para instantes futuros. Estes quadros consistem na coleção de objetos, com estado, para um tempo específico no futuro. Portanto, os estados precisam ser armazenados de tal forma que possibilitem ser exibidos no momento correto. Uma solução é armazená-los em uma fila de estados de objetos. Isso permite que se acesse a esta fila para buscar o estado ideal de cada objeto a ser exibido no tempo  $T$ . A Figura 3.5 demonstra como ocorre a interação com a fila de estados.

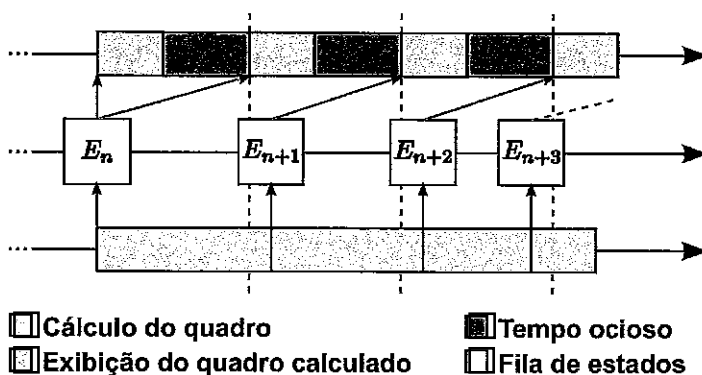


Figura 3.5: As interações com a fila de estados

Como pode ser observado, um estado é inserido na fila no instante em que termina o cálculo da física. Assim, quando  $T_F < T$ , isto significa que não existe um estado

de objeto para o momento de exibição atual. Neste caso, é exibido o último estado armazenado na fila o que pode resultar na perda da continuidade do movimento. Porém, isto não impede que o usuário interaja com os objetos normalmente, pois a exibição não para à espera de um novo quadro.

Quando  $T_F > T$ , a fila de estados possui, pelo menos, um estado para um instante futuro. Como  $\Delta t$  não está associado ao intervalo de exibição, é possível que o estado ideal para  $T$  esteja entre dois estados da fila. Neste caso, é necessária uma interpolação entre os estados a fim de determinar qual será exibido. Assim, a partir de dois estados, é possível determinar novos estados de acordo com a necessidade.

No modelo tradicional, a animação para quando o cálculo da física demora mais do que o intervalo entre as exibições, gerando um atraso em relação ao tempo real. Neste modelo, o tempo  $T_F$  sempre é igual a  $T$ . No modelo proposto, o tempo  $T_F$  é, normalmente, diferente de  $T$ . Neste caso,  $T$  sempre representa o tempo real, e, quando existe um atraso, apenas  $T_F$  está atrasado. O atraso significa que o quadro exibido não corresponde necessariamente ao tempo  $T$ , mesmo que a animação continue em movimento. Contudo, um atraso somente pode ser evitado caso o cálculo do quadro seja concluído rapidamente, ou seja, existe a possibilidade de atrasos tanto no modelo tradicional como no proposto.

O cálculo de estados para instantes futuros tem suas vantagens, conforme mencionado anteriormente. Porém, uma animação física com interação do usuário pode resultar na necessidade de descartar estes estados, pois foram calculados sem considerar as alterações causadas pelo usuário. Uma opção é descartar todos os estados futuros dos objetos ou apenas do objeto participante da interação do usuário. Neste caso, bastaria descartar os estados da fila quando um objeto fosse movido pelo usuário. Esta solução é a mais prática, mas pode resultar em problemas na detecção de colisão. Isto pode ocorrer, por exemplo, caso outro objeto já tenha um estado calculado levando em consideração a configuração anterior da cena.

Considerando esse problema, uma solução proposta é dividir o cenário da simulação em diversas áreas. Com isso, é possível amenizar o problema, pois somente os objetos contidos na área de interação com o usuário devem ter seus estados futuros descartados. No entanto, ao descartar tais estados, pode-se afetar um estado futuro de um objeto em outra área, o que resultaria na necessidade de descartá-los tam-

bém. Portanto, comparam-se os estados calculados anteriormente com os estados calculados após a interação com o usuário para determinar se há a necessidade de descartá-los. Caso seja identificada uma alteração significativa, os estados subsequentes devem ser descartados. Caso contrário, evita-se que mais objetos tenham seus estados descartados.

A partir do momento que ocorre uma interação, o tempo  $T_F$  passa a ser igual ao tempo  $T$ . Caso isto ocorra quando  $T_F < T$ , assume-se que o cenário atualizado pelo usuário corresponde à nova situação que será considerada no próximo cálculo das leis da física.

A divisão em áreas permite também aperfeiçoar a detecção de colisão entre os objetos. Afinal, somente os objetos de uma área serão testados entre si para determinar se houve alguma colisão, diminuindo o domínio para testes. Esta vantagem pode ser melhor observada em cenários com vários objetos espalhados em diversas áreas.

Além disso, é importante manter as áreas independentes umas das outras. Assim, cada uma tem uma propriedade que corresponde ao menor tempo decorrido para um objeto. Esta propriedade possibilita indicar para qual tempo deverá ser computado o próximo estado dos objetos desta área.

Portanto, dependendo das ações ocorridas no cenário, este tempo pode ser diferente em cada área. Por exemplo, a Figura 3.6 demonstra duas situações particulares de interação do usuário. Considerando-se que as quatro áreas tenham estados futuros já calculados: na figura 3.6(a), o objeto  $B$  é adicionado na cena, mas não colide com nenhum outro objeto da área; na figura 3.6(b), o objeto  $C$  é adicionado na cena, no momento seguinte, e altera o estado do objeto  $A$  após colidirem.

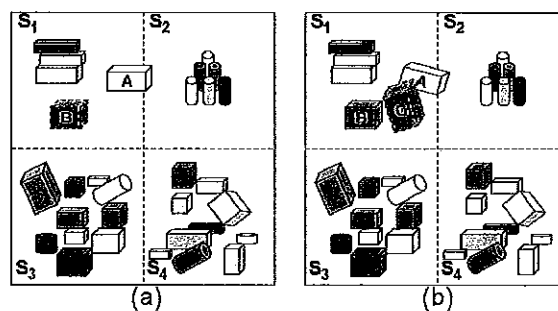


Figura 3.6: Exemplos de interação com usuário.

Para ambas as situações, todos os objetos da área  $S_1$  devem ter seus estados

futuros descartados e são recalculados a partir do momento de interação. Logo, é necessário alterar o tempo desta área para quando ocorreu a interação. As áreas  $S_3$  e  $S_4$  mantêm os estados futuros dos objetos e não precisam ter novos estados calculados, mantendo-se o valor do tempo da área inalterado. Na primeira situação, como não houve colisão com  $A$ , somente  $S_1$  precisa ser recalculada, mesmo com  $A$  pertencendo também à área  $S_2$ . Entretanto, em uma segunda situação, houve uma colisão com  $A$ , que teve seu estado alterado, obrigando os objetos de  $S_2$  a serem recalculados devido à possibilidade de existir uma nova colisão não considerada anteriormente.

Outra vantagem da divisão em áreas é a possibilidade de recalculer os estados somente para uma área específica quando os resultados do tratamento de colisão apresentarem uma imprecisão. Para tanto, as áreas dos objetos envolvidos nesta colisão devem ser marcadas para serem recalculadas com  $\Delta t$  menor. Uma forma de marcar estas áreas é não atualizar o tempo da área, mantendo o tempo em que houve o problema. Além destas áreas, também não se deve atualizar o tempo  $T_F$  com  $T$  neste momento. Portanto, as outras áreas não são processadas por estarem com um tempo maior do que  $T_F$ . Então, todas as áreas somente serão processadas novamente quando os tempos das áreas forem iguais a  $T_F$ . Isso é necessário para evitar erros no tratamento de colisões devido às áreas estarem em tempos muito diferentes.

No próximo capítulo, é descrita a implementação de um protótipo para o modelo proposto neste trabalho. Portanto, são apresentadas algumas bibliotecas necessárias para tal e as alterações necessárias no modelo tradicional para incorporar as idéias do modelo proposto. E, por fim, os algoritmos essenciais para o funcionamento do modelo serão apresentados com mais detalhes.

# Capítulo 4

## Implementação do Modelo

### Proposto

Como o cálculo e a exibição precisam ser executados em paralelo, a implementação do modelo é separada em dois fluxos de execução (*threads*). Para isso, foi utilizada a biblioteca *POSIX threads* (*pthreads*) [36] que permite criá-los. Portanto, um fluxo é responsável pela exibição dos quadros e o controle das ações do usuário e o outro, por calcular as leis da física sobre os objetos da cena.

Conforme mencionado no capítulo anterior, os estados consistem das propriedades físicas para um dado momento, tais como, posição, velocidade, orientação, forças, etc; e das propriedades de exibição do objeto, tais como, a cor do objeto, os vértices, a forma de exibir, etc. Para a exibição, somente duas propriedades físicas são essenciais: a posição e a orientação dos objetos na cena. Porém, outras podem ser necessárias para o cálculo dos estados subsequentes. Por exemplo, com a interação do usuário, faz-se necessário retornar o estado de todos os objetos para o instante da ação, sendo primordial recuperar suas propriedades físicas para manter a animação física coerente com o quadro anterior. A relação entre os dois fluxos de execução com a fila de estados é representada por um esquema de produtor/consumidor [37], como pode ser observado na Figura 4.1.

Para o cálculo das leis da física, são necessários a detecção e o tratamento das colisões. Conforme explicado no item 2.3, existem diversas técnicas utilizadas para a detecção simplificada. Neste protótipo, foram utilizadas três técnicas: *bounding spheres* [38], *Sort and Sweep* [13, 14], conhecida também como *Sweep and Prune* [15],

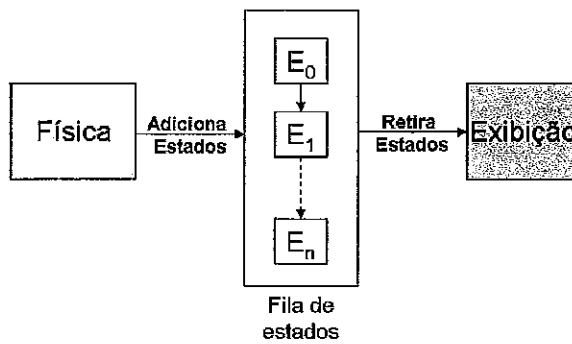


Figura 4.1: Relação entre a física, a exibição e a fila de estados

e uma grade espacial retangular simples.

Com o intuito de melhorar o desempenho da detecção simplificada, optou-se em usar a grade espacial em conjunto com *Sort and Sweep*. A grade limita os objetos que serão ordenados pelo *Sort and Sweep* melhorando o desempenho e diminuindo o número de testes desnecessários. Assim, quando um objeto é adicionado a cena, define-se em que célula da grade o objeto pertence a partir de sua AABB. Sempre que um objeto tem seu estado alterado, a situação dele na grade deve ser atualizada.

Além disso, esta mesma grade espacial é utilizada para determinar em qual área ocorre a ação do usuário e, então, limitar os objetos com estados descartados da fila, conforme descrito em 3.2. Portanto, cada célula deve apresentar as seguintes informações: o menor tempo de cálculo entre todos os objetos e a lista com os objetos pertencentes à célula. Estas informações são atualizadas no momento em que os objetos são inseridos nas células.

As próximas etapas do tratamento de colisão consistem na detecção exata e na resposta à colisão que são baseadas em [10], conforme abordado no item 2.4. A implementação desta técnica segue, basicamente, os seguintes passos:

1. Atualizar a velocidade e a posição de todos os objetos sem considerar as colisões;
2. Detectar as colisões;
3. Retornar os valores iniciais da velocidade e da posição;
4. Atualizar a velocidade considerando as colisões detectadas;
5. Tratar os contatos entre os objetos;



6. Atualizar a posição a partir da velocidade calculada no quarto passo.

Inicialmente, as velocidades e as posições são calculadas sem considerar as possíveis colisões. Com os objetos posicionados na cena, é feita a detecção das colisões e calculadas as forças resultantes destas. No terceiro passo, recuperam-se os dados dos objetos anteriores à execução do primeiro passo. E com os objetos na situação inicial, é computado o resultado final da atuação das leis da física, levando em consideração as forças determinadas durante o segundo passo. Para o modelo proposto, é necessário adaptar essa estrutura para aplicar as técnicas mencionadas. Portanto, o fluxo principal de execução do cálculo das leis da física é o seguinte:

1. Garantir que todos os objetos tenham o estado para o tempo  $T_F - \Delta t$  que corresponde ao tempo imediatamente anterior ao tempo de simulação corrente  $T_F$ . Para tanto, os objetos devem ter seus estados recuperados da fila para o tempo  $T_F - \Delta t$ ;
2. Atualizar a velocidade e a posição para todos os objetos da cena, sem considerar as colisões;
3. Atualizar as situações dos objetos na grade;
4. Determinar as células que devem ser computadas comparando o tempo de simulação da célula com  $T_F$ ;
5. Fazer a detecção dos pares de colisão das células de forma individual garantindo que não haja repetição;
6. Fazer a detecção de colisão exata para todos os pares identificados;
7. Recuperar as posições e as velocidades anteriores de todos os objetos;
8. Atualizar a velocidade e a posição para todos os objetos da cena, considerando as colisões;
9. Verificar se os objetos das células calculadas, pertencentes a mais de uma célula, tiveram o seu estado alterado em relação ao estado já armazenado para o tempo considerado. Caso o estado seja diferente, todas as células interceptadas pelo objeto devem ser consideradas no próximo cálculo;

10. Salvar os novos estados dos objetos nas respectivas filas de estados.

Quando um objeto é criado, o estado inicial é inserido na respectiva fila, garantindo-se que haja um estado a ser exibido. Antes da exibição de um novo quadro, recupera-se na fila de estados de cada objeto a melhor situação possível para o tempo  $T$ .

## 4.1 Algoritmos

Neste tópico, são apresentados os algoritmos necessários para implementação do modelo proposto. Para tanto, consideram-se as seguintes definições:

1.  $n$  é o número de objetos na cena;
2.  $E_i$  é a fila de estados do objeto  $i$ , em que,  $i \in [0..n - 1]$ . Cada estado  $E_i[j]$  corresponde a um instante de tempo  $t(E_i[j])$  em que as propriedades físicas do objeto  $i$  foram computadas. Os estados são inseridos em ordem crescente de tempo e  $|E_i|$  é o número de estados na fila  $E_i$ . Então,  $t(E_i[j]) < t(E_i[j + 1])$  para  $j \in [0..|E_i| - 1]$ .

Durante todo o tempo da animação física, são gerados e inseridos novos estados na fila de estados. Assim, para cada  $\Delta t$ , um novo estado do objeto  $i$  é inserido em  $E_i$ . Durante o processo de exibição, o objeto  $i$  tem suas propriedades retornadas por  $BuscaEstado(i, t)$ , em que  $t$  é o tempo para qual o estado é necessário. Basicamente, essa função é responsável por retornar o estado do objeto  $i$  da fila de estados para um tempo específico. Com o intuito de descrever o algoritmo, considere os seguintes termos:

1.  $\varepsilon$  representa um pequeno intervalo de tempo usado para comparar dois instantes de tempo. Então,  $t_1$  e  $t_2$  são considerados iguais se  $t_1 \in [t_2 - \varepsilon, t_2 + \varepsilon]$ ;
2.  $L$  é a lista de objetos, em que  $L_i$  é o  $i$ -ésimo objeto, tal que,  $i \in [0..n - 1]$ ;
3.  $Interpola(E_i[j], E_i[k], t)$  retorna um estado interpolado para o tempo  $t$  baseado em dois estados conhecidos,  $E_i[j]$  e  $E_i[k]$ . As posições são interpoladas linearmente, enquanto as orientações são computadas usando a interpolação linear

esférica, conhecida como SLERP (*spherical linear interpolation*), apresentada por Ken Shoemake[39].

Com essas definições, o algoritmo 1 descreve como o estado é retornado para um tempo específico. O estado retornado pela função pode ser: um estado armazenado próximo ao tempo solicitado  $t$ , o último estado armazenado na fila quando o tempo da física  $T_F$  é maior que o tempo atual de exibição  $T$ , ou um estado interpolado entre dois estados da fila quando os casos anteriores não forem atendidos. Inicialmente, verifica-se a existência de mais do que um estado na fila, pois caso  $T_F \leq T$ , a simulação física está atrasada em relação à exibição e somente existe um estado possível a ser retornado para exibir. Se existem dois ou mais estados na fila, procura-se pelo estado correspondente ao tempo  $t$  utilizando  $\varepsilon$  para comparar com os tempos dos estados. Caso não seja encontrado um estado na fila que atenda a este critério é verificada a existência de outro para um período maior ao tempo  $t$ , e é realizada a interpolação entre o primeiro estado com tempo maior e o estado anterior a esse para o tempo  $t$ . Esse caso é identificado quando  $T_F > T$ .

---

**Algoritmo 1:** BuscaEstado

---

**Input:**  $i$ : índice de um objeto;  
**Input:**  $t$ : o tempo específico;  
**Output:** estado do objeto  $L_i$  para o tempo  $t$ ;  
**begin**  
    **if**  $t - \varepsilon \leq t(E_i[0]) \leq t + \varepsilon$  **then**  
        | **return**  $E_i[0]$   
    **for**  $j \leftarrow 1$  **to**  $|E_i| - 1$  **do**  
        | **if**  $t - \varepsilon \leq t(E_i[j]) \leq t + \varepsilon$  **then**  
            | **return**  $E_i[j]$   
        | **if**  $t(E_i[j - 1]) + \varepsilon \leq t \leq t(E_i[j]) - \varepsilon$  **then**  
            | **return**  $Interpola(E_i[j - 1], E_i[j], t)$   
    **return**  $E_i[|E_i| - 1]$   
**end**

---

Já o algoritmo 2 demonstra a implementação do fluxo principal da simulação da física para o tempo  $T_F$ . Inicialmente, garante-se que todos os objetos da cena estão com estados para o tempo  $T_F - \Delta t$  que corresponde ao último estado calculado antes de  $T_F$ . O próximo passo consiste em posicionar todos os objetos considerando-se  $\Delta t$  e em determinar as novas células em que se encontram. A partir disso, verificam-se as células que devem ter novos estados calculados para os objetos. Isto é possível

ao verificar-se que o tempo da célula é menor do que  $T_F$ . Posteriormente, aplica-se uma detecção simplificada somente para os objetos contidos dentro das células para identificar quais são os pares de colisão. Estes são inseridos em uma lista única que não permite repetições e, em seguida, é aplicada uma detecção exata para determinar quais os pares colidiram realmente. Os seguintes conceitos são utilizados no algoritmo 2:

1.  $G$  é a grade espacial da cena.  $G_m$  é a lista de objetos de uma célula  $m$ ;
2.  $P$  é a lista de pares de colisão;
3. *CalculaEstados* calcula novos estados  $E'_i$  para o objeto  $L_i$  que apresentou colisão. Então,  $\forall L_i | L_i \in \{P\}$ , existe um  $E'_i$ ;
4. *AtualizaEstados(c)* atualiza a lista de estados dos objetos da célula  $c$  e marca as células que devem ser computadas.
5. *Ativa(c)* verifica se a célula  $c$  está ativa. Em caso verdadeiro, significa que, pelo menos, um objeto inserido em  $c$  está em movimento. Caso contrário, todos os objetos de  $c$  estão parados.

---

**Algoritmo 2:** CalculaFisica

---

```

Input:  $L_c$ : lista de índices de células
begin
   $T_F \leftarrow T_F + \Delta t$ 
  for  $i \leftarrow 0$  to  $|L| - 1$  do
     $L_i \leftarrow BuscaEstado(i, T_F - \Delta t)$ 
    AtualizaVel( $i$ )
    AtualizaPos( $i$ )
    AtualizaGrade( $i$ )
  for  $m \leftarrow 0$  to  $|G| - 1$  do
    if  $t(m) < T_F$  Ativa( $m$ ) then
       $L_c.push(m)$ 
      DetecSimplificada( $L(m)$ )
    DetecExata( $t$ )
    CalculaEstados
  for  $k \leftarrow 0$  to  $|L_c| - 1$  do
    AtualizaEstados( $k$ )
end

```

---

Portanto, as células consideradas no tratamento de colisões e, por conseguinte, no cálculo de um novo estado, devem atender a duas proposições: o tempo da célula  $t(c)$  deve ser menor que  $T_F$  e a célula deve estar ativa, ou seja, deve possuir, pelo menos, um objeto em movimento. A primeira proposição somente não é atendida quando ocorre uma interação com o usuário ou um novo objeto é inserido na cena. Nestes casos, a célula da ação tem o tempo atualizado com  $T$  diferente das outras células que são atualizadas com  $T_F$ . No fim da simulação corrente, o tempo  $T_F$  é atualizado com  $T$ . Assim, estas atualizações distintas de tempo caracterizam quais as células não precisam ser calculadas na próxima simulação da física. Já a segunda proposição, tem como objetivo melhorar o desempenho e reduzir o número de objetos testados no tratamento de colisões. Assim, todas as células são consideradas não ativas inicialmente. No entanto, uma célula é considerada ativa quando um objeto em movimento for inserido nela. Isso pode ocorrer quando um novo objeto for inserido na cena ou com a mudança de célula de um objeto em movimento.

Considera-se que o tempo máximo de simulação do objeto  $i$  é  $t(i)$ . Assim, devido à possibilidade das células possuírem tempos de simulação distintos, sempre que um objeto  $i$  for inserido em uma célula  $c$ , compara-se o seu tempo  $t(i)$  com o da célula  $t(c)$ , e se  $t(i) < t(c)$ , então  $t(c)$  é atualizado com  $t(i)$ , ou seja,  $t(c) \leftarrow t(i)$ . Isso é necessário, pois não se deve fazer o tratamento de colisões entre objetos com tempos diferentes. Com esta atualização de tempo, indica-se que na próxima simulação os objetos da célula precisam ter novos estados calculados a partir do tempo  $t(i)$  devido às possíveis novas colisões com a presença de  $i$ .

Após o tratamento de colisões, os novos estados são inseridos na fila de estados de cada objeto. Porém, verifica-se antes se estes novos estados são diferentes dos calculados anteriormente para o mesmo tempo. Considere, pela figura 4.2, que demonstra a inserção do objeto  $B$  na cena. Este objeto é inserido na célula  $C_1$  e a princípio somente os objetos desta célula precisam ter os estados da fila descartados. Contudo, caso ocorra uma colisão com o objeto  $A$  e o novo estado seja diferente do antigo para o mesmo tempo, os objetos da célula  $C_2$  também devem ter os estados futuros descartados e novos estados armazenados na fila considerando a nova posição de  $A$ .

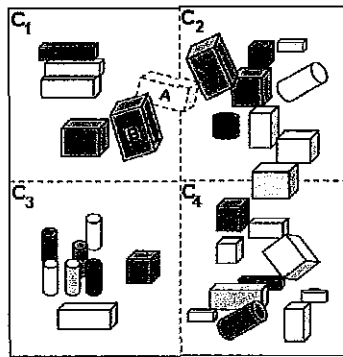


Figura 4.2: Exemplo de inserção de um objeto na cena

Tanto a atualização da fila de estados como esta verificação são apresentadas no algoritmo 3 descrito a seguir:

---

**Algoritmo 3:** AtualizaEstados

---

**Input:**  $c$ : índice de uma célula da grade  
**Input:**  $L(c)$ : lista de objetos pertencentes a célula  $c$   
**Input:**  $C_i$ : lista de células do objeto  $i$   
**begin**  
  **for**  $i \leftarrow 0$  **to**  $|L(c)| - 1$  **do**  
     $E_i'' \leftarrow BuscaEstado(i, T_F)$   
    **if**  $E_i'' \geq T_F$  **then**  
      **if**  $E_i'' \neq E_i'$  **then**  
         $E_i[j].erase(E_i''.index, |E_i| - 1)$   
         $E_i.push(E_i')$   
         $t(i) \leftarrow T_F$   
      **else**  
         $E_i.push(E_i')$   
         $t(i) \leftarrow T_F$   
    **end**  
**end**

---

Na seção 3.2, foi descrita a solução para o problema do desperdício. Essa solução implica em um  $\Delta t$  que possibilite melhorar a utilização dos recursos no cálculo de estados futuros. Porém, pode causar o problema de imprecisão em determinados casos. O único modo de solucionar este problema é reduzir o valor de  $\Delta t$  para estes casos. Entretanto, a imprecisão somente pode ser identificada após determinar os pares de colisão, o que resulta em um processamento desnecessário das etapas anteriores.

Portanto, identifica-se a possibilidade da existência deste problema testando a profundidade da colisão entre os objetos do par. Ao encontrar uma imprecisão, o tempo de simulação dos objetos deste par não é atualizado com  $T_F$ , mantendo-os como se não tivessem sido processados na simulação corrente. Além disso, ao final da simulação,  $T_F$  deve ser atualizado com o tempo da simulação anterior, ou seja,  $T_F \leftarrow T_F - \Delta t$ . Para o próximo cálculo da física,  $\Delta t$  é reduzido a fim de eliminar a imprecisão identificada. Assim, somente a célula com os objetos do par de colisão identificado anteriormente é processada.

# Capítulo 5

## Resultados

Um protótipo do sistema descrito foi implementado na linguagem C++ usando, para desenhar a cena, a biblioteca OpenGL [40, 41]. Usou-se também a biblioteca OpenGL Utility Toolkit (GLUT) [42, 43] que permite simplificar a programação com OpenGL e abstrair o sistema operacional, permitindo a portabilidade do código para a criação de aplicativos em diversas plataformas.

O protótipo foi usado para realizar diversos experimentos em um computador equipado com 2Gb de memória principal, um processador Intel Core 2 Duo 2.13 GHz, e uma placa gráfica NVIDIA GeForce 7600 GS.

Com a biblioteca GLUT, a exibição é realizada em um tempo aproximadamente constante e pequeno, e é possível configurar a aplicação para exibir os quadros em um intervalo de tempo específico. Assim, a aplicação foi configurada para exibir a uma taxa de 30 quadros por segundo aproximadamente em todos os experimentos realizados. Além disso, o número máximo de estados armazenados para cada objeto foi limitado em 100 estados e  $\Delta t$  utilizado no cálculo da física foi de 0,030 segundos.

Com esses parâmetros iniciais definidos, foram realizados quatro experimentos a fim de testar o funcionamento e o desempenho do modelo proposto neste trabalho. O primeiro e o segundo visam avaliar se o cálculo de estados futuros melhora o desempenho da aplicação de modo a manter a continuidade do movimento. A diferença entre os dois experimentos reside na complexidade das malhas dos objetos utilizados. O terceiro experimento testa a idéia de separação de áreas no tratamento de colisões a partir da interação do usuário. E por fim, o quarto experimento demonstra o funcionamento do retorno no tempo para um caso em que o cálculo de



colisões não obteve um resultado plausível.

Assim, o primeiro experimento consiste em uma cena simples com paralelepípedos rígidos (255), os quais representam o movimento da queda de dominós em espiral, como observado na figura 5.1.

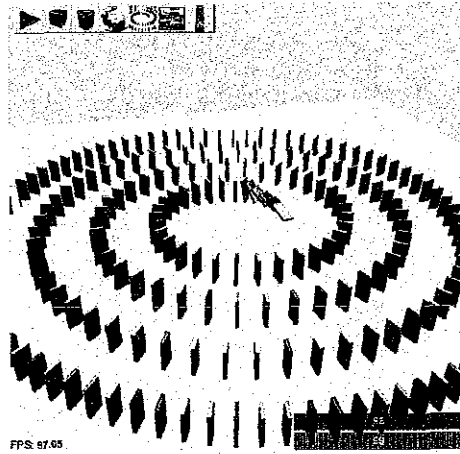


Figura 5.1: Uma cena simples com 255 paralelepípedos representando dominós dispostos em espiral.

Inicialmente, nota-se um crescimento rápido do tempo da física devido ao avanço no tempo para calcular estados futuros e armazená-los na fila. Este preenchimento pode ser observado na figura 5.2.

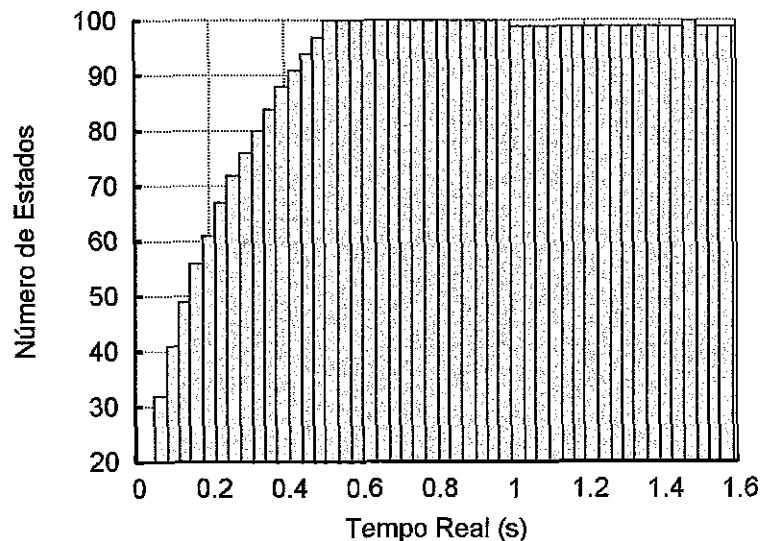


Figura 5.2: Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo do dominó.

Já a figura 5.3 demonstra a evolução do tempo de exibição e da física em relação ao tempo real. Com isso, nota-se que o tempo da física está sempre a frente do

tempo de exibição confirmando que os estados armazenados são para tempos futuros à exibição. Ao comparar estas duas figuras, observa-se que o crescimento do tempo da física estabiliza-se e os gráficos passam a ser praticamente paralelos após as filas de estados estarem cheias. Neste caso, o crescimento não é maior por falta de espaço na fila para armazenar novos estados. Assim, quando um estado é retirado da fila, um novo é calculado imediatamente em seguida mantendo a diferença entre os tempos de exibição e da física praticamente constante.

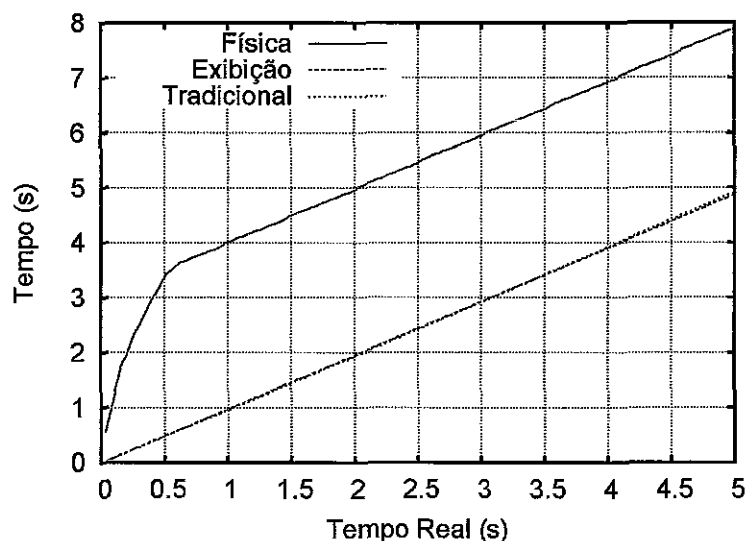


Figura 5.3: Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo do dominó.

Além disso, outro ponto importante observado na figura 5.3 é a evolução do gráfico de exibição em relação ao tempo real. Como esperado, o cálculo da física desassociado da exibição não influencia o tempo de exibição possibilitando uma animação em tempo real. Nesta mesma figura, é demonstrado o comportamento do experimento para o modelo tradicional com intuito de comparação. Para este caso, a animação também pode ser considerada em tempo real, pois devido à simplicidade dos objetos utilizados, o cálculo da física não atrasa a exibição.

Com intuito de explorar melhor o funcionamento da fila de estados, o segundo experimento é realizado com objetos de malha geométrica mais complexa do que o primeiro exemplo. Assim, a configuração da cena para este experimento é apresentada na figura 5.4.

Neste caso, a fila de estados demorou mais tempo para ser totalmente preenchida, pois não foi possível avançar muito no tempo devido ao tempo para calcular a física



Figura 5.4: Exemplo com malhas complexas.

para estes objetos, conforme a figura 5.5. Apesar disso, foi possível calcular diversos estados à frente da exibição aproveitando, principalmente, os momentos em que o cálculo era mais simples. Isto pode ser observado no início do gráfico quando o número de estados na fila era próximo de 40 e depois diminuiu para menos de 30. Esta diminuição ocorreu devido ao tempo maior para calcular os estados para o momento em que os objetos colidiram com o chão e, em seguida, entre si.

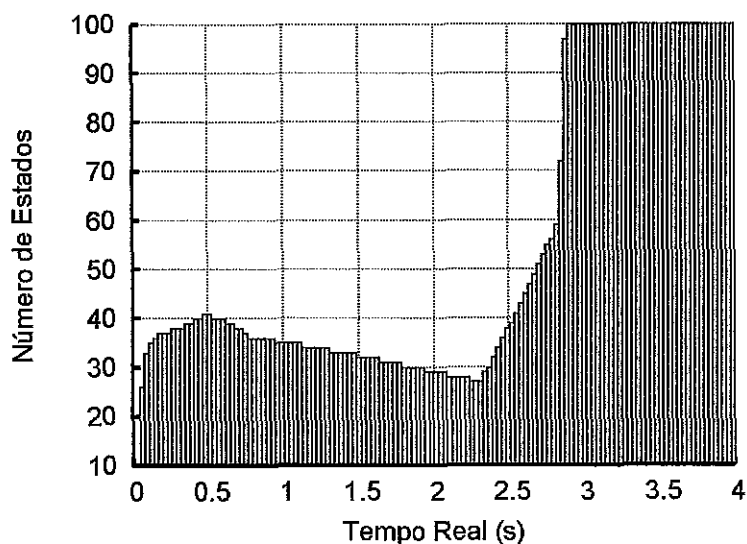


Figura 5.5: Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de malhas mais complexas.

Contudo, o avanço no tempo garantido inicialmente possibilitou manter a animação em tempo real como pode ser comprovado pelos gráficos da figura 5.6. Nota-se que o crescimento do gráfico da física diminuiu em um determinado momento. Porém, esta diminuição não possibilitou que o tempo de exibição fosse maior que o tempo

da física garantindo assim que a animação física transcorresse em tempo real.

No modelo tradicional, a execução deste experimento não obteve um bom resultado como no caso dos dominós. Na figura 5.6, é apresentado o gráfico de tempo da física/exibição para o modelo tradicional (os gráficos da física e da exibição são aproximadamente iguais). Nota-se que houve um atraso considerável em relação ao tempo real a partir do momento em que ocorreu a colisão entre os objetos. Assim, a animação física deste experimento não foi exibida em tempo real para o modelo tradicional.

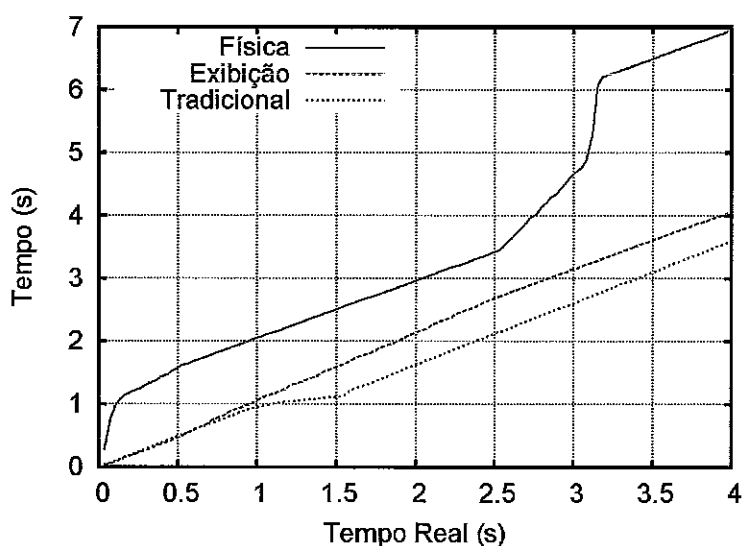


Figura 5.6: Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo de malhas complexas.

Após testar a eficiência da fila de estados, realizou-se um experimento para verificar a funcionalidade da divisão de áreas de ação. O cenário escolhido foi o do dominó por tratar-se de uma animação contínua e longa, conforme a figura 5.1. Com isso, é possível adicionar objetos na cena enquanto os dominós continuam em movimento. Assim, a idéia é observar o comportamento da fila de estados e do tempo da física quando os objetos são adicionados na cena. Para tanto, optou-se por adicionar um objeto aos 10 segundos de animação. Na figura 5.7, é possível observar o momento em que foi adicionado o objeto e a exibição do último estado calculado para cada objeto. Estes estados correspondem aos objetos transparentes da figura.

Lembrando que o tempo da física passa a ser igual ao tempo de exibição do momento em que o objeto foi adicionado à cena, pode-se observar pela figura 5.8,

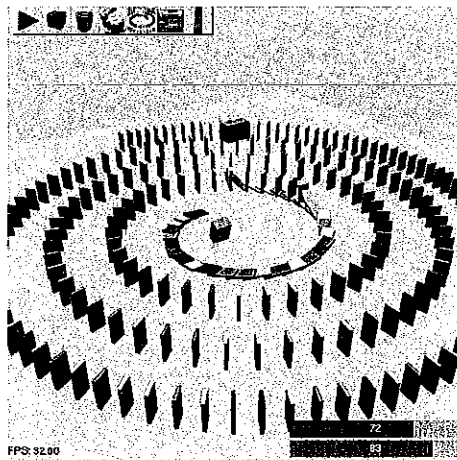


Figura 5.7: Exemplo de dominós com interação do usuário. A barra vermelha representa o número mínimo de estados em fila e a barra azul indica o número máximo de estados em fila.

que o número máximo de estados começa a diminuir no momento em que um novo objeto é adicionado na cena. Isto ocorre porque os objetos com tempo de simulação maior do que o tempo da física deixa de ter estados computados, mas não pararam de ter estados exibidos e retirados da fila.

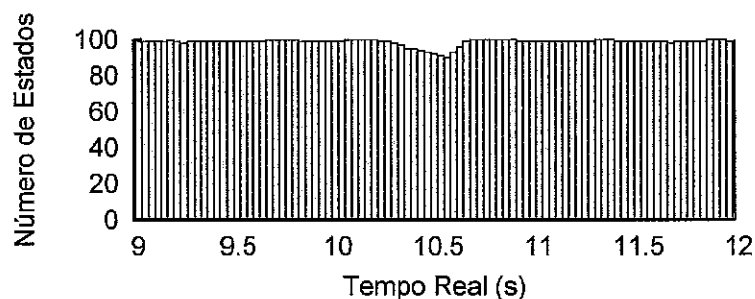


Figura 5.8: Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de dominós com interação do usuário.

Ao mesmo tempo, são computados estados futuros para o novo objeto, preenchendo assim a sua fila de estados e aumentando o tempo da física. Quando os outros objetos tiverem seus tempos de simulação inferiores ao tempo da física novamente, estes passam a ter novos estados computados. Com isso, o número máximo de estados aumenta até a fila estar completamente cheia. Esta variação do tempo

da física pode ser observado na figura 5.9

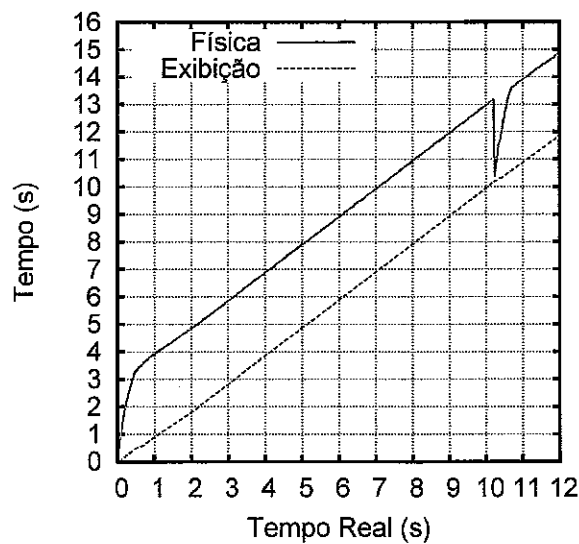


Figura 5.9: Gráfico do tempo de exibição e da física em relação ao tempo real para o exemplo de dominós com interação do usuário.

O último experimento consiste em testar o retorno no tempo quando o cálculo da física não tiver um resultado plausível. Este problema foi identificado em deslocamentos verticais com o valor de  $\Delta t$  utilizado, isto é, 0,030 segundos. Portanto, a configuração da cena escolhida é uma pilha de 20 paralelepípedos de acordo com a figura 5.10.

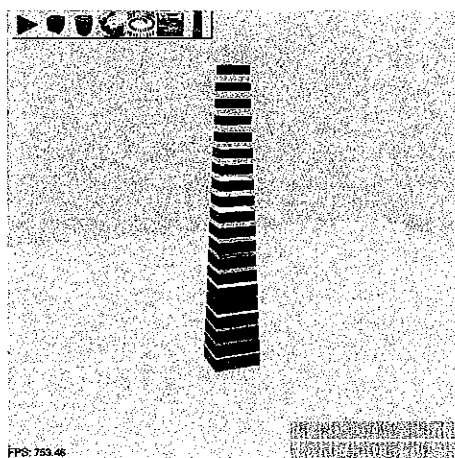


Figura 5.10: Exemplo de uma pilha de paralelepípedos.

Inicialmente, o experimento foi executado sem o retorno no tempo para verificar o comportamento. Observou-se que realmente o cálculo da física não resolve todas as colisões, pois alguns objetos permaneceram interpenetrados havendo a perda de equilíbrio e, conseqüentemente, a queda dos objetos da pilha. Ao executar o

experimento com o retorno no tempo, o cálculo da física teve um bom resultado, resolvendo as colisões de forma estável. Estas duas situações podem ser observadas na figura 5.11.

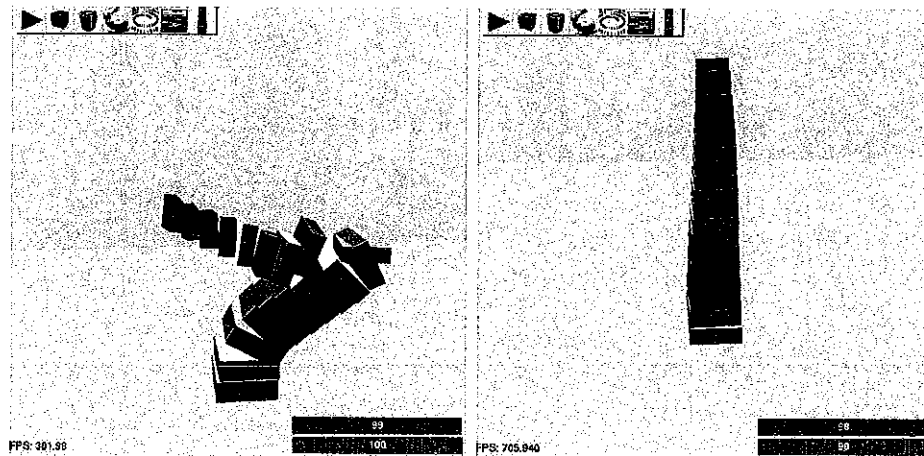


Figura 5.11: A pilha da esquerda sem retorno no tempo; A da direita com retorno no tempo.

Apesar da necessidade de retornar no tempo, não houve problemas em computar estados suficientes para preencher toda a fila, como observado na figura 5.12. Já na figura 5.13, são apresentados os gráficos de evolução do tempo da física e de exibição em relação ao tempo real.

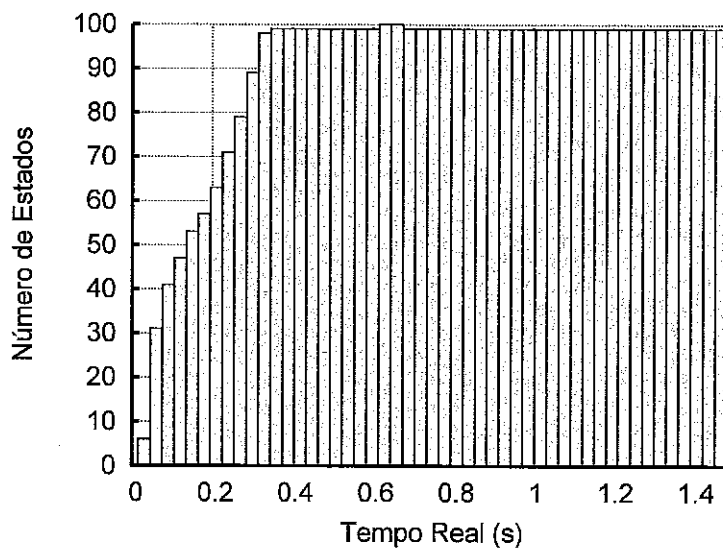


Figura 5.12: Gráfico do número de itens na fila de estados em cada momento de exibição para o exemplo de pilha de paralelepípedos

Além disso, o gráfico de variação de  $\Delta t$  em relação ao tempo real também é apresentado. Com este gráfico, é possível notar a adaptação do valor de  $\Delta t$  de

acordo com a necessidade. Portanto, o valor inicial manteve-se constante enquanto os objetos não estavam tão próximos um dos outros.

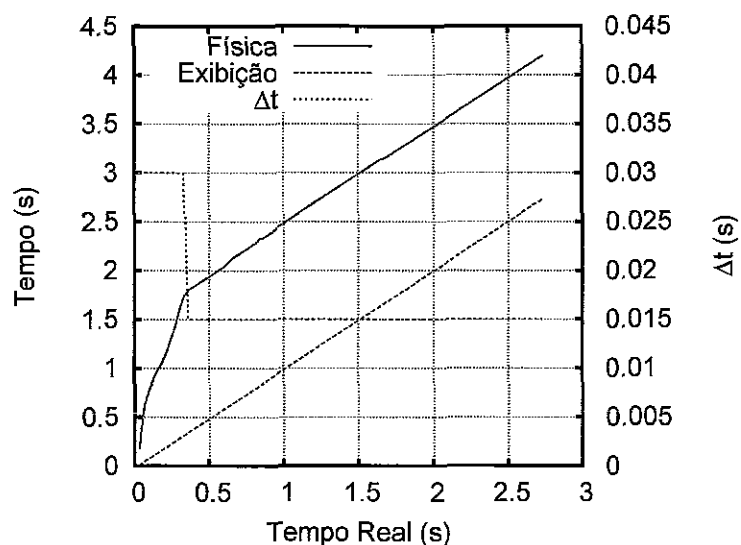


Figura 5.13: Gráfico do tempo de exibição, da física e de  $\Delta t$  utilizado em relação ao tempo real para o exemplo da pilha.

A partir do momento em que as colisões foram detectadas e os resultados do cálculo da física não estavam satisfatórios, o valor de  $\Delta t$  foi reduzido para a metade do valor inicial. Com este novo valor, os cálculos da física tiveram um bom resultado, conforme constatado na pilha da direita da figura 5.11.



# Capítulo 6

## Conclusões

Nesta dissertação, descreveu-se uma abordagem para a animação de corpos rígidos na qual o cálculo dos parâmetros físicos é desacoplado da exibição em si. Adicionalmente, é possível armazenar quadros a serem exibidos futuramente de forma que um eventual atraso no cálculo da física não seja sentido pelo observador. Além disso, também é apresentado um esquema de decomposição temporal da cena que viabiliza o aproveitamento de partes pré-computadas da animação mesmo em face da obsolescência de outras partes devido, por exemplo, a uma alteração interativa da cena. Um protótipo incorporando estas idéias foi implementado e experimentos foram realizados de forma a demonstrar a exequibilidade da proposta.

A partir desses experimentos, foi possível constatar a melhora na animação, pois não houve a interrupção nos movimentos dos objetos devido à espera pelo fim do cálculo da física. E, com a possibilidade de armazenar os quadros para momentos futuros, a abordagem proposta tendeu a produzir uma animação mais suave e com maior precisão em comparação com o modelo tradicional. Assim, apresentou-se resultados em tempo real para situações que não eram possíveis no modelo tradicional. Além disso, o processamento assíncrono tornou possível tratar o cálculo físico com a precisão necessária, por exemplo, usando passos de integração variáveis no caso da pilha de paralelepípedos.

Portanto, com a análise dos resultados dos testes, concluiu-se que o avanço no tempo em situações simples permite um tempo maior para calcular situações mais complexas evitando o atraso da animação em relação ao tempo real. Já a separação em áreas de ação diminui o desperdício de processamento devido à restrição dos

objetos considerados na interação com o usuário e, conseqüentemente, possibilita a manutenção de estados futuros já calculados para os objetos de outras áreas.

Contudo, uma forma de melhorar o desempenho é a utilização de técnicas mais eficientes de detecção de colisão simplificada e de resposta a colisões. Além disso, pode-se aprimorar o procedimento quando o usuário interage com os objetos. O fato de calcular constantemente novos estados e descartar os estados futuros quando o objeto é movimentado pelo usuário provoca um processamento desnecessário e dificulta a manutenção do sincronismo entre a exibição e a fila de estados, aumentando as chances de gerar um erro no cálculo da física.

Além dessas melhorias, uma opção para um trabalho futuro é aumentar o número de fluxos de execução. Com isso, aproveita-se a separação em áreas já implementada para dividir o tratamento de colisões em diversos fluxos de execução o que, provavelmente, permitirá a exibição em tempo real de uma cena com um número maior de objetos e uma melhor utilização dos recursos disponíveis.

# Referências Bibliográficas

- [1] BATTAIOLA, A. L., FEIJÓ, B., DE G. DOMINGUES, R., et al., “Desenvolvimento de Jogos em Computadores e Celulares”, *Revista de Informática Teórica e Aplicada. Edição Especial: Computação Gráfica e Processamento de Imagens*, v. VIII, n. 2, 2001.
- [2] COUMANS, E., “Bullet Physics Library”, <http://www.bulletphysics.com>, 2009.
- [3] NVIDIA, “NVIDIA PhysX”, [http://www.nvidia.com/object/nvidia\\_physx.html](http://www.nvidia.com/object/nvidia_physx.html), 2009.
- [4] BARAFF, D., “Interactive Simulation of Solid Rigid Bodies”, *IEEE Comput. Graph. Appl.*, v. 15, n. 3, pp. 63–75, 1995.
- [5] HAHN, J. K., “Realistic animation of rigid bodies”. In: *In SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pp. 299–308, ACM Press: New York, NY, USA, 1988.
- [6] HECKER, C., “Rigid body dynamics”, <http://www.d6.com/users/checker/dynamics.htm>, 1998.
- [7] MIRTICH, B. V., *Impulse-based dynamic simulation of rigid body systems*, Ph.D. Thesis, University of California at Berkeley, 1996.
- [8] BARAFF, D., “Fast contact force computation for nonpenetrating rigid bodies”, *In SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 23–34, 1994.
- [9] VAN DEN BERGEN, G., “Efficient collision detection of complex deformable models using AABB trees”, *J. Graph. Tools*, v. 2, n. 4, pp. 1–13, 1997.

- [10] GUENDELMAN, E., BRIDSON, R., FEDKIW, R., “Nonconvex Rigid Bodies with Stacking”, *ACM Trans. Graphics (SIGGRAPH Proc.)*, v. 22, n. 3, pp. 871–878, 2003.
- [11] WITKIN, A., BARAFF, D., “Physically Based Modeling: Principles and Practice”, *Online Siggraph '97 Course notes*, 1997, <http://www.cs.cmu.edu/~baraff/sigcourse>.
- [12] GOTTSCHALK, S., LIN, M. C., MANOCHA, D., “OBBTree: a hierarchical structure for rapid interference detection”. In: *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, ACM: New York, NY, USA, 1996.
- [13] BARAFF, D., *Dynamic Simulation of Non-Penetrating Rigid Bodies*, Ph.D. Thesis, Cornell University, March 1992.
- [14] ERICSON, C., “Real-time Collision Detection”, chap. 7, pp. 285–348, *Morgan Kaufmann series in interactive 3D technology*, Amsterdam: Elsevier, 2005.
- [15] COHEN, J. D., LIN, M. C., MANOCHA, D., et al., “I-COLLIDE: an Interactive and Exact Collision Detection System for Large-Scale Environments”, *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pp. 189–196, April 1995.
- [16] GARCIA-ALONSO, A., SERRANO, N., FLAQUER, J., “Solving the Collision Detection Problem”, *IEEE Computer Graphics and Applications*, v. 14, n. 3, pp. 36–43, 1994.
- [17] GANOVELLI, F., DINGLIANA, J., “BucketTree: Improving collision detection between deformable objects”. In: *In SCCG2000 Spring Conf. on Comp. Graphics*, pp. 156–163, 2000.
- [18] ZHANG, D., YUEN, M. M. F., “Collision Detection for Clothed Human Animation”. In: *PG '00: Proceedings of the 8th Pacific Conference on Computer Graphics and Applications*, p. 328, IEEE Computer Society: Washington, DC, USA, 2000.

- [19] BANDI, S., THALMANN, D., “An Adaptive Spatial Subdivision of the Object Space for Fast Collision Detection of Animated Rigid Bodies”, *Computer Graphics Forum*, v. 14, n. 3, pp. 259–270, 1995.
- [20] LUQUE, R. G., COMBA, J. L. D., FREITAS, C. M. D. S., “Broad-phase collision detection using semi-adjusting BSP-trees”. In: *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pp. 179–186, ACM: New York, NY, USA, 2005.
- [21] TELLER, S. J., SÉQUIN, C. H., “Visibility preprocessing for interactive walkthroughs”, *SIGGRAPH Comput. Graph.*, v. 25, n. 4, pp. 61–70, 1991.
- [22] TESCHNER, M., HEIDELBERGER, B., MÜLLER, M., et al., “Optimized spatial hashing for collision detection of deformable objects”. pp. 47–54, 2003.
- [23] LEFEBVRE, S., HOPPE, H., “Perfect spatial hashing”. In: *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pp. 579–588, ACM: New York, NY, USA, 2006.
- [24] SHINYA, M., FORGUE, M., “Interference detection through rasterization”, *Journal of Visualization and Computer Animation*, v. 2, pp. 131–134, 1991.
- [25] HEIDELBERGER, B., TESCHNER, M., GROSS, M., “Detection of collisions and self-collisions using image-space techniques”. In: *Journal of WSCG*, pp. 145–152, 2004.
- [26] GOVINDARAJU, N. K., REDON, S., LIN, M. C., et al., “CULLIDE: interactive collision detection between complex models in large environments using graphics hardware”. In: *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 25–32, Eurographics Association: Aire-la-Ville, Switzerland, Switzerland, 2003.
- [27] GOVINDARAJU, N. K., KNOTT, D., JAIN, N., et al., “Interactive collision detection between deformable models using chromatic decomposition”, *ACM Trans. Graph.*, v. 24, n. 3, pp. 991–999, 2005.

- [28] DAVID KNOTT, M., *CInDeR Collision and Interference detection in real time using graphics hardware*, Master's Thesis, UBC, 2003.
- [29] TESCHNER, M., HEIDELBERGER, B., MANOCHA, D., et al., "Collision Handling in Dynamic Simulation Environments: The Evolution of Graphics: Where to next?" *In Eurographics Tutorial number 2*, 2005.
- [30] HAN-YOUNG JANG, TAEKSANG JEONG, J. H., "Gpu-based image-space approach to collision detection among closed objects", *In PG 2006: Proceedings of the 2006 Pacific Conference on Computer Graphics and Applications*, pp. 242–251, Oct 2006.
- [31] HAN-YOUNG JANG, TAEKSANG JEONG, J. H., "Image-space collision detection through alternate surface peeling". In: *Advances in Visual Computing*, pp. 66–75, Springer Berlin / Heidelberg, November 2007.
- [32] HARADA, T., "Real-Time Rigid Body Simulation on GPUs". In: *GPU Gems 3*, pp. 25–32, Addison-Wesley Professional, 2007.
- [33] BARAFF, D., "Curved surfaces and coherence for non-penetrating rigid body simulation", *SIGGRAPH Comput. Graph.*, v. 24, n. 4, pp. 19–28, 1990.
- [34] POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., et al., "Interactive manipulation of rigid body simulations". In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 209–217, ACM Press/Addison-Wesley Publishing Co.: New York, NY, USA, 2000.
- [35] COLTHEART, M., "The persistences of vision", *Philos Trans R Soc Lond B Biol Sci*, pp. 57–69, July 1981.
- [36] BUTENHOF, D. R., *Programming with POSIX threads. Addison-Wesley professional computing series*, Addison-Wesley, 1997.
- [37] TANENBAUM, A. S., "Modern Operating Systems", 3rd ed., chap. 2, Pearson Prentice Hall: Upper Saddle River, N.J., 2008.

- [38] DOPERTCHOUK, O., “Simple Bounding-Sphere Collision Detection”, <http://www.gamedev.net/reference/articles/article1234.asp>, November 2000.
- [39] SHOEMAKE, K., “Animating rotation with quaternion curves”, *ACM SIGGRAPH Computer Graphics*, v. 19, pp. 245–254, July 1985.
- [40] BOARD, O. A. R., SHREINER, D., *OpenGL reference manual: The official reference document to OpenGL, version 1.4*. 4th ed. Addison-Wesley, 2004.
- [41] BOARD, O. A. R., SHREINER, D., WOO, M., et al., *OpenGL programming guide : the official guide to learning OpenGL, version 2*. 5th ed. Addison-Wesley, 2006.
- [42] KILGARD, M. J., *Programming OpenGL for the X Window System*. 5th ed. Addison-Wesley Developers Press, 1996.
- [43] KILGARD, M. J., *The OpenGL Utility Toolkit (GLUT) Programming Interface: API Version 3*. Silicon Graphics, Inc., November 1996.
- [44] SOLOMON, C., *Enchanted Drawings: The History of Animation*. 1st ed. New York: Knopf, 1989.
- [45] CHABUKSWAR, R., LAKE, A. T., LEE, M. R., “Multi-threaded Rendering and Physics Simulation”, *Intel® Software Solutions Group (SSG)*, February 2005.
- [46] BOND, A., STRUNK, O., GASINSKI, A., “Physics simulation apparatus and method”, , n. 20060149516, July 2006.
- [47] TESCHNER, M., HEIDELBERGER, B., MÜLLER, M., et al., “Optimized Spatial Hashing for Collision Detection of Deformable Objects”, *In Proc. of Vision, Modeling, Visualization VMV*, pp. 47–54, November 2003.
- [48] LASSETER, J., “Principles of Traditional Animation Applied to 3D Computer Animation”, *Computer Graphics*, pp. 35–44, July 1987.

- [49] BURMYK, N., WEIN, M., “Computer Generated Keyframe Animation”, *Journal of the SMVrE 80*, pp. 149–153, March 1971.
- [50] BURMYK, N., WEIN, M., “Interactive skeleton techniques for enhancing motion dynamics in key frame animation”, *Communications of the ACM*, v. 19, n. 10, pp. 564–569, October 1976.
- [51] BACIU, G., WONG, W. S. K., “Image-Based Techniques in a Hybrid Collision Detector”, *IEEE Transactions on Visualization and Computer Graphics*, v. 9, n. 2, pp. 254–271, 2003.
- [52] BACIU, G., WONG, W. S.-K., SUN, H., “RECODE: an image-based collision detection algorithm”, *Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on*, pp. 125–133, Oct 1998.