Universidade Federal de Pernambuco
Graduação em Ciência da Computação

Centro de Informática

SIMULAÇÃO FÍSICA DE CORPOS DEFORMÁVEIS EM TEMPO REAL

Trabalho de Graduação

Aluno: Mozart William Santos Almeida (mwsa@cin.ufpe.br)
Orientadora: Judith Kelner (jk@cin.ufpe.br)
Co-orientadora: Veronica Teichrieb (vt@cin.ufpe.br)

Recife, 21 de Janeiro de 2008

Resumo

A simulação de objetos deformáveis vem demonstrando ser de grande importância para o aumento do realismo em aplicações interativas que, de alguma maneira, precisam retratar fielmente o comportamento físico de seus elementos. Este é um dos principais objetivos de algumas aplicações de realidade virtual e aumentada, bem como de alguns jogos eletrônicos modernos. Desta forma, o principal desafio desta área consiste em desenvolver técnicas que cumpram de forma satisfatória a demanda entre precisão numérica e velocidade de execução. Este Trabalho de Graduação contextualiza os principais ramos de atuação da simulação física na atualidade, apresenta o estado da arte no ramo de simulação física de corpos deformáveis em geral, porém dando maior enfoque nas técnicas de tempo real. Além disso, propõe o desenvolvimento de duas dessas técnicas, a *Position Based Dynamics* proposta por Müller *et al.* e a técnica de objetos deformáveis estruturados proposta por Desbrun *et al.* realizando assim, uma análise comparativa entre elas baseado em suas principais características e comparando com os resultados originais apresentados.

Palavras-chave: Simulação física, Corpos deformáveis, Tempo real, Restrições

Agradecimentos

É com muita satisfação que chego ao final de mais essa fase. Gostaria de agradecer a todas as pessoas que direta ou indiretamente – mesmo que tenham sido como “simples” observadores – contribuíram para que este dia finalmente chegasse.

Primeiramente, agradeço à possibilidade de fazer parte da força sutil onipresente, que faz parte de mim, define o meu Eu e que me dá, agora e sempre, sustentação e equilíbrio para continuar buscando o caminho do meio.

Muito obrigado aos meus amados pais, Zumiro e Carmelita, que além de terem me dado todo o apoio que se pode ter desde o início, têm tentado compreender, da melhor forma possível, as idéias caóticas que habitam a cabeça deste ser que mais (pensa que) pensa do que fala. Muito obrigado!

Agradeço também ao meu querido irmão Marcell, que é uma das pessoas que consegue me proporcionar momentos extremamente agradáveis, e apesar de também conseguir o contrário com perfeição, eu ainda o tenho como um modelo a ser seguido... de certa forma. Valeu, bro!

Obrigado também aos grandes apoiadores da minha causa, o Sr. João Soares e a Sra. Iracy, que onde quer que ela esteja certamente está feliz por este dia ter chegado. E, Cidinha, que também tem me dado suporte (e me suportado) por todo este tempo. Obrigado!

Aos que contribuíram diretamente para a execução deste trabalho, utilizando sua ampla experiência nos mais variados campos. Gabriel, Joma, Mouse, Rodrigo, Pedro, Guilherme, Curupa e todo o pessoal do GPRT/GRVM, que de alguma forma – mesmo que apenas pelo fato de existirem e observarem – contribuíram para o fim desta jornada, meus agradecimentos.

Professoras Judith e Veronica, que me orientaram ao longo desta jornada e me deram a oportunidade de aumentar minhas capacidades em certas tarefas e alcançar objetivos que, em outros tempos, acredito que seriam muito difíceis de realizar. Muito Obrigado!

Muito obrigado também aos amigos-irmãos de faculdade, aos integrantes da equipe Default, do CRK!, do Default++, de Jogos, de Comunicação sem Fio, de Inglês, de TALC, de PGP, de APS, das Infras e de todas as outras que propiciaram momentos engraçados, polêmicos e, por vezes, irritantes, que hoje em dia servem como (ótimas) piadas para os breves momentos em que nos encontramos em lugares aleatórios.

Roberto, Cesar, Chico, João Paulo, Raphael, Daliton, Antonio, Domingos e todo o pessoal da turma 2003-2, que certamente de alguma forma contribuíram para este dia e suas realizações. Muito Obrigado!

Mais uma vez, muito obrigado e perdão pelos possíveis transtornos causados.

Sumário

1. Introdução

O desenvolvimento de aplicações interativas requer a modelagem dos objetos que compõem a cena em diversos níveis. A modelagem geométrica descreve a forma e aparência destes objetos, a modelagem cinemática determina a localização 3D dos objetos no espaço bem como seu movimento, enquanto que a modelagem comportamental é responsável por associar comportamentos independentes do usuário aos objetos [1]. Nesta última categoria inclui-se a modelagem física, que integra o objeto às suas características físicas, como peso, inércia, aspereza de sua superfície, etc.

Basicamente, em relação à simulação de elementos físicos, podem-se simular fisicamente dois tipos de objetos: rígidos ou deformáveis. Estes, ao interagirem com algum outro corpo durante a simulação, por exemplo, sofrerão algum tipo de deformação, de acordo com as características do choque, dos corpos envolvidos na interação e das características do ambiente que os cerca.

Diferentemente, os corpos rígidos representam elementos rigidamente ideais, ou seja, não deverão sofrer nenhum tipo de deformação sob qualquer tipo de interação. A Figura 1 exemplifica a interação entre estes dois tipos de corpos.

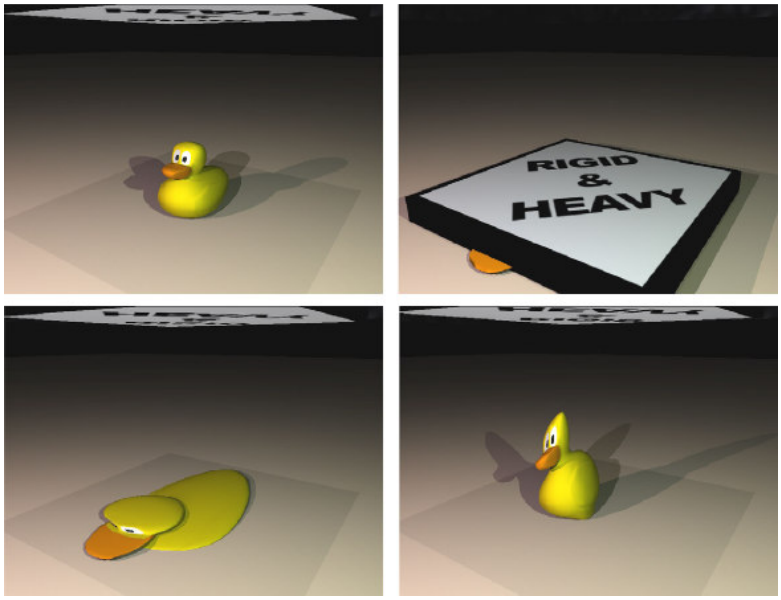

Figura 1. Exemplo de interação entre um objeto deformável e um corpo rígido [2].

Bastante conhecida pela sua grande interdisciplinaridade, a simulação de corpos deformáveis não é uma área recente, principalmente quando relacionada a áreas do conhecimento como Engenharia Mecânica ou Ciência dos Materiais. Nestas áreas o

principal objetivo é simular o comportamento preciso de certos corpos quando submetidos a certas condições específicas. Para isto, muita matemática é necessária, abordando principalmente temas como dinâmica newtoniana, computação numérica, cálculo diferencial, dentre outros [8].

Aliada a computação gráfica, entretanto, este tipo de simulação vem obtendo atenção da comunidade, sempre trazendo bastantes resultados significativos a cada nova característica ou técnica desenvolvida. Há aproximadamente vinte anos, em discussões a respeito de animação 3D, Lasseter [3], juntamente ao trabalho pioneiro de Terzopoulos [4], que descreve um método de simulação física baseada na teoria da elasticidade, foram os precursores da inclusão de modelos com comportamento físico realista em animações.

Desde então, novas técnicas vêm sendo desenvolvidas para uma utilização mais realista de objetos maleáveis em filmes e aplicações científicas – com pouca ou nenhuma interatividade – procurando reduzir apenas os erros provenientes da discretização dos modelos inerentemente contínuos [5].

Em áreas que exigem interatividade, como Realidade Virtual (RV), Realidade Aumentada (RA) e Jogos, principalmente suas versões mais recentes, as técnicas utilizadas devem ser mais simples, visto que necessitam de respostas às interações do usuário no momento da aplicação do estímulo.

A Figura 2 ilustra um treinador virtual para cirurgia endoscópica que possibilita a movimentação de órgãos, suturações, etc [6].

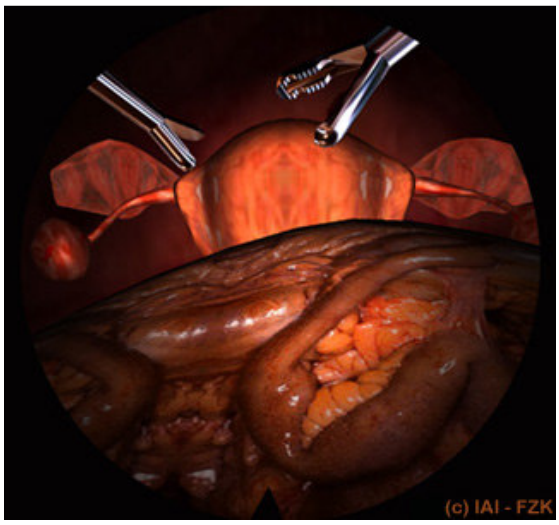

Figura 2. Treinador cirúrgico utilizando modelos deformáveis.

A Figura 3 exibe o recente jogo Crysis™ que possui uma forma bastante realista de interação com os ambientes virtuais através de simulação física, incluindo elementos deformáveis e quebráveis [7].

Figura 3. Interação fisicamente realista em tempo real com o ambiente no jogo Crysis™.

Como resposta à crescente demanda por realismo nestas áreas, aliada à rápida evolução da infra-estrutura de *hardware* necessária para este tipo de aplicação, a utilização em tempo real de corpos deformáveis auxilia no aumento dos níveis de realismo, uma vez que nem todos os objetos encontrados na natureza podem ser coerentemente representados apenas por corpos rígidos.

As técnicas de simulação física em geral mais utilizadas podem ser classificadas em três principais categorias: as baseadas em força, em impulso e em posição. As baseadas em força necessitam de uma integração de segunda ordem sobre o tempo a fim de atualizar as posições das partículas [8]. As técnicas baseadas em impulso calculam os novos valores das posições a partir da manipulação de velocidades [9]. As baseadas em posição manipulam diretamente as posições dos vértices, evitando alguns cálculos de integração, possibilitando um maior controle sobre os elementos manipulados [10] [11].

1.1. Objetivo

O presente Trabalho de Graduação tem por objetivo o estudo de técnicas e ferramentas de simulação física, com foco principal em corpos deformáveis. A partir disto, um estudo mais aprofundado de duas dessas técnicas será apresentado, bem como sua implementação abrangendo – em sua totalidade ou parcialmente – algumas de suas principais características.

As técnicas a serem aprofundadas serão a baseada em posição *Position Based Dynamics* [10], proposta por Müller *et al.*, e o modelo de animação interativa proposto por Desbrun *et al.* [12], pelo fato de possuírem relativo baixo custo de execução, tornando-as viáveis à utilização em aplicações de tempo real.

Desta forma, poderão ser traçadas conclusões a respeito da viabilidade de utilização de cada uma dessas técnicas no âmbito de aplicações interativas, assim como uma análise comparativa tanto do seu desempenho quanto do compromisso existente entre suas principais vantagens e desvantagens.

De acordo com os resultados e análises obtidos, possíveis modificações ou melhorias poderão ser propostas.

1.2. Organização do documento

O próximo capítulo aborda alguns conceitos teóricos básicos necessários para facilitar o entendimento dos algoritmos que serão detalhados ao longo do texto. O capítulo 3 contextualiza simulação física de forma mais generalizada e fornece exemplos do tipo de ferramentas existentes para facilitar o desenvolvimento de aplicações interativas.

O capítulo 4 se restringe à simulação de objetos deformáveis, demonstrando algumas técnicas de tempo real. No capítulo 5 são descritos os estudos de caso, bem como o detalhamento de cada técnica implementada, seus resultados e as principais dificuldades encontradas. No capítulo 6 são feitas as considerações finais esclarecendo as principais contribuições geradas por este trabalho, e finalizando com a sugestão de possíveis trabalhos futuros nesta área.

2. Conceitos relacionados

Os conceitos matemáticos que embasam o desenvolvimento das técnicas apresentadas neste trabalho são conhecidos por sua extensão e complexidade. Porém, para o entendimento das técnicas detalhadas no capítulo 5, um conhecimento básico mínimo é requerido. Desta forma, os próximos tópicos serão abordados de uma forma sucinta.

A utilização dos dois primeiros assuntos abordados neste capítulo, a saber, diferenciação parcial e gradiente, têm sua utilização principal na obtenção de direções e taxas de variação na movimentação dos vértices dos objetos. A última seção dá uma visão geral das principais formas de integração sobre o tempo, utilizadas para simulação física de objetos deformáveis.

2.1. Diferenciação parcial

A derivada parcial de uma função de mais de uma variável, equações bastante comuns em cálculos físicos, é a derivada relativa a uma de suas variáveis independentes, considerando as demais como constantes na derivação [13].

O exemplo a seguir demonstra a notação mais utilizada para indicar que a diferenciação foi executada com respeito a apenas uma das variáveis independentes da função, independentemente dos nomes das variáveis utilizadas.

Seja a seguinte função, que representa a lei dos gases ideais, que é a equação do estado de um gás ideal hipotético:

$$P(V,T) = n\frac{RT}{V}. \tag{1}$$

Dado que n é o número de mols do gás (constante), R é a constante dos gases universais, T é a temperatura (variável) e V o volume do sistema (variável).

Notadamente, esta função possui duas variáveis independentes, T e V. Se a temperatura varia com o tempo enquanto o volume permanece constante, a taxa de variação (derivada) da pressão com respeito à temperatura $\frac{\partial P}{\partial T}$ será:

$$\frac{\partial P}{\partial T} = n\frac{R}{V}. \tag{2}$$

Esta nova equação representa a derivada relativa à T, considerando V como uma constante. Caso o volume varie enquanto a temperatura permanecer constante, a taxa de variação $\frac{\partial P}{\partial V}$ será:

$$\frac{\partial P}{\partial V} = -n\frac{RT}{V^2}. \tag{3}$$

Neste caso, o processo é o mesmo da equação anterior, apenas considerando T como constante.

2.2. Gradiente

Em equações que descrevem o movimento de partículas, por exemplo, há a necessidade de se conhecer a taxa de variação numa determinada direção utilizando o conceito de derivada direcional [13]. Por exemplo, a derivada direcional $D_{\vec{u}}f(x,y,z)$ de uma função qualquer $f(x,y,z)$ na direção de um vetor unitário $\vec{u} = (a,b,c)$ é igual à soma dos produtos das derivadas parciais da função f por cada coordenada do vetor $\vec{u}$, respectivamente. Ou seja:

$$D_{\vec{u}}f(x,y,z) = \frac{\partial f}{\partial x}a + \frac{\partial f}{\partial y}b + \frac{\partial f}{\partial z}c. \tag{4}$$

Esta equação pode ser escrita como o produto escalar entre um vetor formado pelas derivadas parciais de f e o vetor unitário $\vec{u}$, que indica a direção da variação:

$$D_{\vec{u}}f(x,y,z) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right) \cdot (a,b,c). \tag{5}$$

O primeiro vetor desta operação é conhecido como o gradiente de f ou o vetor gradiente de f e pode ser representado da seguinte maneira:

$$grad(f) = \nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right). \tag{6}$$

O valor máximo de $D_{\vec{u}}f(x,y,z)$, ou seja, a taxa máxima de variação da função $f(x,y,z)$ é igual a $||\nabla f(x,y,z)||$, ocorrendo na direção de $\nabla f(x,y,z)$.

Então, se $\nabla f(x,y,z) \neq 0$, o gradiente é perpendicular à superfície (considerando um sistema tridimensional), caso $f(x,y,z) = 0$ [13][14].

2.3. Integração sobre o tempo

Várias são as alternativas para realizar a integração sobre o tempo, a fim de encontrar as novas posições dos vértices e gerar uma animação coerente. Os

esquemas de integração sobre o tempo são primariamente classificados com relação à estabilidade e precisão [8].

O esquema mais simples é a integração Euleriana explícita, que é assim conhecido pelo fato de que resulta em fórmulas explícitas para os novos valores do próximo *timestep*. Apesar de sua facilidade de implementação, este esquema impõe algumas restrições para manter a estabilidade num nível aceitável, como a necessidade de cada *timestep* ser inversamente proporcional à raiz quadrada do *stiffness* do sistema [12]. *Stiffness* refere-se à resistência à elongação dos corpos elásticos. [2] exemplifica uma das principais desvantagens de se utilizar o esquema explícito.

$$v_i^{n+1} = v_i^n + F_i^n \frac{dt}{m} \tag{7}$$

$$x_i^{n+1} = x_i^n + v_i^{n+1} dt \tag{8}$$

Nas equações acima, nota-se que forças no tempo t_n contribuem para encontrar as velocidades no tempo t_{n+1}, ou seja, forças que ainda vão ser calculadas podem mover os pontos dos objetos para lugares indesejados.

Uma modificação no esquema explícito tornou-se a principal alternativa a ser utilizada: a integração implícita Euleriana, que ao invés de fornecer os resultados da próxima iteração baseando-se nas forças relativas à iteração atual, utiliza-se das forças da próxima iteração para encontrar os resultados da iteração atual. Este esquema também é conhecido como esquema Euleriano reverso (*backward Euler scheme*).

Métodos de integração de mais alto nível como *Runge-Kutta* [15] também podem ser utilizados para este fim, conferindo até maiores níveis de estabilidade que os anteriormente citados. Porém, segundo [12], para aplicações de tempo real que necessitam de muitas colisões este esquema se mostra inviável, visto que é mais adequado para resultados com movimentos suaves e contínuos.

3. Simulação física

Desde o momento em que surgiram sistemas computacionais suficientemente eficientes para executar os cálculos complexos envolvidos na simulação física, várias aplicações para este tipo de simulação surgiram como em animações para a indústria do cinema ou na indústria automobilística, por exemplo.

A Figura 4 ilustra um simulador de testes de colisão (*crash-tests*) virtual de alta fidelidade, que promove a redução de custos, por fazer como que menos veículos reais sejam necessários para executar estes tipos de testes [16].

Figura 4. Comparação de resultados entre *crash-test* virtual e real.

A partir de então, e com a melhoria crescente dos computadores, tanto em termos de velocidade de execução quanto em capacidade de armazenamento, novas áreas do conhecimento foram sendo favorecidas com o surgimento de aplicações utilizando esta nova forma de simulação, dentre as quais se enquadram a Realidade Virtual e Aumentada, além dos Jogos eletrônicos modernos. Como exemplificado nas figuras 2 e 3, no capítulo 1.

Com a evidente divisão entre simulação precisa (*offline*) e em tempo real, surgiram vários modelos de simulação, que procuravam satisfazer a um compromisso que também existe em áreas correlatas, como a Computação Gráfica, por exemplo. Tal compromisso é a relação entre a precisão da simulação e a velocidade de execução [17].

As aplicações que prezam mais pela precisão numérica, como as citadas no início deste capítulo e ilustradas pela Figura 4, não têm como principal foco a interatividade, mas sim a análise ou visualização dos resultados após um tempo –

geralmente longo – de cálculos bem mais complexos que discretizam os modelos físicos essencialmente contínuos.

Por outro lado, para aplicações que requerem interatividade com o usuário, velocidade de execução e baixas latências são extremamente importantes, visto que o principal foco deste tipo de aplicação é fazer com que o usuário perceba o mundo virtual como se fosse real, e este é por essência interativo. Assim, os cálculos tendem a ser mais simples, tornando a precisão física um fator secundário, desde que os objetos sendo simulados possuam um comportamento visualmente aceitável.

Atualmente, com a popularização das máquinas capazes de executar os cálculos de simulação interativamente, popularizou-se também o fato de que muitos jogos eletrônicos estão utilizando simulação física para adicionar mais realismo a seus efeitos.

Desta forma, a fim de facilitar o desenvolvimento de aplicações que utilizem simulação física em tempo real, nos últimos anos tem-se difundido bastante a utilização dos motores físicos para aplicações de tempo real que, assim como os motores gráficos, procuram abstrair as operações e estruturas relacionadas, que evitam que o desenvolvedor tenha de se preocupar com sua implementação, se preocupando apenas com o desenvolvimento de sua solução, seja ela um simulador de Realidade Virtual ou um Jogo, por exemplo.

A próxima seção fornece uma visão geral dos principais motores físicos de tempo real existentes, incluindo suas principais características.

3.1. Motores físicos

Um motor físico é um *software* que simula modelos físicos newtonianos, a fim de tentar simular o comportamento de certos elementos sob determinadas circunstâncias. Assim como há métodos de simulação física de alta precisão e de tempo real, também existem motores físicos que abstraem estes modelos, tanto os precisos, quanto os de tempo real [18]. Nesta seção será dado foco a motores de tempo real.

Com a crescente difusão deste tipo de simulação em aplicações de tempo real, muitos tipos de motores surgiram, para satisfazer as mais variadas necessidades. Desta forma, há motores de código aberto (*open source*), comerciais e livres com código fechado. No decorrer desta seção, será dada uma visão geral destes vários motores existentes.

3.1.1. Open Dynamics Engine

O *Open Dynamics Engine* ODE [15] é um motor físico simples, de código aberto, que se propõe a simular corpos rígidos e suas articulações, conhecidas como juntas.

Ele também provê um sistema de detecção de colisões que, diferentemente de alguns simuladores, é baseado em restrições, tornando a simulação mais estável. Outra forma de tratar colisão em geral é utilizando molas virtuais, porém este modelo gera alguns artefatos indesejados à simulação.

O ODE contém um variado suporte a juntas, como a dobradiça (*hinge*), joelho (*ball-and-socket*) ou pistão (*slider*), por exemplo. A Figura 5 ilustra estes tipos de juntas. A *ball-and-socket* é mais adequada para simular elementos que rotacionam de forma similar a um ombro, por exemplo. A junta do tipo *hinge*, como seu nome implica, simula dobradiças, visto que os objetos podem se movimentar apenas com relação a um eixo. A junta *slider* permite que os objetos se movimentem ao longo de um mesmo eixo, como um pistão ou amortecedor [19].

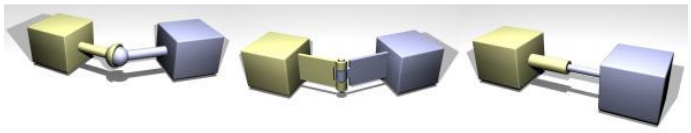

Figura 5. Exemplos de juntas disponíveis no ODE. Da esquerda para a direita: *ball-and-socket*, *hinge*, *slider*.

3.1.2. Ageia PhysX

Anteriormente conhecida como *NovodeX Physics SDK*, o *PhysX* [20] da *Ageia* é um dos mais poderosos motores físicos para aplicações em tempo real existentes. Anteriormente, ele era um motor de desenvolvimento pago, para aplicações comerciais.

Para aplicações acadêmicas ou com fins educativos, sua licença era gratuita. Recentemente houve mudanças, de forma que para qualquer tipo de aplicação o *PhysX* agora é gratuito, porém seu código é totalmente fechado, sendo necessário pagar, caso demande alguma alteração em seu núcleo.

O PhysX é muito utilizado para jogos, pois possui suporte a todo o ciclo de desenvolvimento de uma aplicação deste tipo, como ilustrado na Figura 6. O suporte engloba *plugins* para exportação de objetos físicos a partir de ferramentas de modelagem, como o 3D Studio Max [21] ou o Maya [22], além de um depurador visual remoto (*Visual Remote Debugger* – VRD), que permite a depuração das aplicações

interativamente, direto na plataforma onde será executada, como no *Microsoft Xbox 360*, *Playstation 3* ou num PC. A Figura 6 ilustra estes componentes e como eles se relacionam.

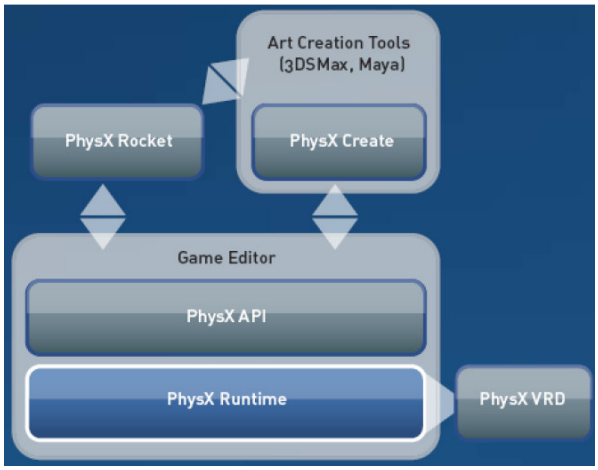

Figura 6. Principais componentes do *PhysX*.

Além de uma complexa dinâmica de corpos rígidos e detecção de colisão, o *PhysX* possui suporte a um grande número de elementos físicos, como juntas e molas, que possibilitam a criação de elementos mais complexos, como a movimentação realista de personagens, a simulação de motores e a possibilidade de manipular objetos no mundo virtual.

Um sistema de partículas com comportamento físico também é suportado, o que faz com que elementos como fogo, fumaça ou neblina possam interagir realisticamente com outros aspectos do ambiente, como fumaça contida num compartimento, por exemplo; a fumaça sobe até o topo do compartimento até preencher toda a região superior, podendo vazar e ser levada pelo vento.

O *PhysX* é um dos poucos motores de tempo real que implementam a simulação de fluidos volumétricos realistas, que interagem com quaisquer outros elementos físicos adicionados à cena. Além disso, tecidos e objetos deformáveis volumétricos também podem ser simulados em tempo real, a fim de melhorar o realismo de cenas em que objetos como cortinas, plantas, roupas, objetos de borracha ou esponjosos estejam presentes. A Figura 7 ilustra a simulação de um tecido sendo rasgado pelo peso de um cubo anexado.

Figura 7. Demonstração de um tecido rasgando devido ao peso do cubo.

Fluidos, tecidos e objetos deformáveis, por utilizarem algoritmos de simulação mais complexos, impulsionaram o surgimento de pesquisas na área de *hardware* dedicado para física, assim como ocorre com os processadores gráficos (*Graphics Processing Units* – GPUs).

Desta forma, no *PhysX* estes três elementos possuem algumas restrições de execução, sendo melhor executados com a presença da recém desenvolvida *Ageia Physics Processing Unit* (PPU), que melhora o desempenho de qualquer elemento físico do motor e conseqüentemente das aplicações que façam uso dela (Figura 8) [20].

Figura 8. Ageia PhysX PPU.

3.1.3. Havok Physics

O *Havok* [23] é um motor físico que faz parte de uma suíte de ferramentas que fornece suporte principalmente ao desenvolvimento de jogos para as mais diversas plataformas. Possui licença estritamente comercial, ou seja, para desenvolver aplicações físicas utilizando *Havok*, é necessário comprar o *kit* de desenvolvimento.

Segundo os desenvolvedores, o motor possui suporte a um modelo de detecção de colisão diferenciado, que não utiliza física discreta [24], eliminando problemas relativos a objetos que se movem muito rápido.

Assim como o *PhysX*, também possui suporte a ferramentas de modelagem e possui um depurador visual a fim de detectar e corrigir erros durante a execução da aplicação.

O *Havok* também possui elementos de modelos de simulação complexa, de forma que, para executar em tempo real adequadamente, se faz necessário utilizar algum tipo de recurso que melhore a execução. Esta melhoria vem através da utilização das GPUs (Figura 9) como um co-processador dedicado para a simulação mais complexa.

Figura 9. Exemplo de placa gráfica utilizada como co-processador físico no *Havok*.

4. Objetos deformáveis

Em aplicações de RV, RA e jogos a utilização apenas de corpos rígidos para aumentar o realismo não satisfaz. Isto se deve ao fato de que o comportamento dos corpos rígidos é apenas uma aproximação do comportamento físico dos objetos, ou seja, nem todos os elementos encontrados na natureza possuem rigidez ideal. Há uma parcela significativa de elementos na natureza que ao serem submetidos à certa força, se deformam e não se quebram, como ocorre com os corpos rígidos.

Por exemplo, um objeto composto de certo tipo de metal maleável como ouro, por exemplo, pode ser eficientemente simulado utilizando-se dinâmica de corpos rígidos caso seja necessário pouca ou nenhuma colisão específica. Porém, se é necessário que esta entidade sofra algum tipo de impacto mais forte, para continuar simulando realisticamente seu comportamento, é necessário que se simule a deformação que este corpo sofrerá, devido às características maleáveis do material que o compõe [25].

Desenvolver aplicações que possuam elementos com este nível de realismo comportamental, geralmente resulta na utilização de equações diferenciais relativamente complexas e/ou a resolução de sistemas lineares grandes. Este tipo de abordagem é utilizado quando são necessários resultados mais precisos do que eficientes computacionalmente. Neste caso, o poder computacional exigido é muito alto, restringindo a utilização das simulações a sistemas não-interativos.

Porém, novas técnicas têm sido desenvolvidas para, tanto tornar mais eficientes as aplicações complexas e precisas [4] e [29], quanto abrir novas possibilidades para o desenvolvimento de novas técnicas bem mais eficientes, conseqüentemente menos precisas, porém fisicamente plausíveis e capazes de executar adequadamente em tempo real.

Os modelos de maior destaque em animação e simulação física de objetos deformáveis tridimensionais se enquadram em duas categorias principais: os Modelos de Elemento Finito (*Finite Element Method* – FEM) e os baseados em massa-mola. Nealen *et al.* e Gibson *et al.* [8][26] fornecem excelentes *surveys* da área, apresentando estas principais abordagens e algumas de suas variações.

Várias são as classificações dadas às técnicas utilizadas para representar objetos deformáveis. Segundo [26], uma das classificações mais amplas é entre a

utilização de princípios físicos e a utilização apenas de geometria para calcular formas e movimentos dos objetos.

Um exemplo deste último tipo de modelagem de deformação física, como os que serão descritos a seguir, podem ser os que são detalhados em [27], que embora não utilizem princípios físicos, são perfeitamente capazes de prover resultados fisicamente corretos.

O principal problema com estes modelos é que a manipulação do formato de determinados modelos pode se tornar exageradamente complexa, visto que, em geral, a manipulação dos pontos de controle não é tão intuitiva, como ilustra a Figura 10.

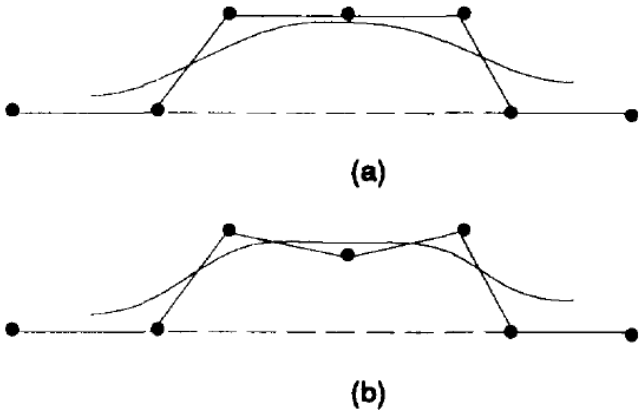

Figura 10. Utilização de *Free-Form Deformation* numa curva. (a) mostra como fica a curva caso os pontos fiquem alinhados horizontalmente. (b) mostra como deve ficar a configuração dos pontos para criar uma linha horizontal [24].

Hsu *et al.*[28] propõe a redução dos problemas de quem necessita utilizar esta abordagem, através da criação de um modelo que permita que o usuário modifique o formato dos modelos através da manipulação direta dos seus vértices.

Ao longo deste capítulo, serão descritas técnicas que representam o estado da arte da pesquisa em simulação física de objetos deformáveis baseados em conceitos físicos, apresentando sumariamente suas principais características e algumas de suas principais aplicações.

A classificação adotada neste Trabalho de Graduação foi em relação à forma de simulação, ou seja, se é necessário um pré-processamento antes do resultado final (simulação *offline*) ou se um usuário pode interagir com a cena durante a simulação (tempo real). Como o presente trabalho é baseado em duas técnicas de simulação em tempo real, foi dada uma maior ênfase a este tipo de simulação.

4.1. Simulação *offline*

FEM são utilizados para encontrar aproximações para funções contínuas que satisfaçam alguma expressão de equilíbrio. Utilizando esta abordagem, modelos inerentemente contínuos são divididos em nós discretos e é calculada uma função que resolva a equação de equilíbrio para cada elemento [26].

Assim, sua utilização depende fortemente da capacidade computacional disponível, tornando-se bastante difícil sua utilização em aplicações de tempo real, pelo principal fato de que seus vetores de força, massas e matrizes envolvidas devem ser recalculados a cada deformação do objeto [26].

Grande parte das aplicações deste tipo de cálculos é encontrada em simuladores em que resultados precisos são indispensáveis, como em animações de alta qualidade para a indústria do cinema [29], planejadores cirúrgicos [30] – que atualmente precisam ser executados de forma *offline* pelo nível de precisão necessário – ou aplicações do ramo automobilístico [31] [16], por exemplo.

Além do FEM, há uma ampla utilização de seus principais derivados, como o Método dos Volumes Finitos (*Finite Volume Method* – FVM), que possui uma maneira mais simples e intuitiva de integrar equações de movimento. Desta forma, segundo [32], seus conceitos podem ser tão simples de entender quanto os sistemas massa-mola, embora ofereçam um nível de complexidade computacional similar ao do FEM.

Terzopoulos *et al.* em seu trabalho pioneiro [4], simula materiais deformáveis, incluindo efeitos de elasticidade, viscoelasticidade, plasticidade e fratura. Neste trabalho, ele sugere a possibilidade de utilizar FEM para tal, porém ele preferiu utilizar outro método de discretização, o Método das Diferenças Finitas (*Finite Difference Method* – FDM). Esta variação do FEM é também mais simples de implementar, porém possui algumas desvantagens relacionadas ao tratamento de malhas regulares, destacadas em [8].

4.2. Simulação em tempo real

O avanço do desenvolvimento da infra-estrutura de *hardware* aliado à demanda por realismo em aplicações interativas tem favorecido o surgimento de modelos mais simples em relação aos apresentados anteriormente que, apesar de menos precisos, oferecem a capacidade de execução em tempo real.

O modelo mais simples e intuitivo para simulação de deformação em tempo real é o modelo massa-mola. Nele, ao invés de utilizar um modelo contínuo e discretizá-lo para executar a simulação como no FEM ou seus derivados, o processo já se inicia discreto [8].

Kharevych e Khan [33], detalham a implementação de um sistema massa-mola simples, incluindo detecção de colisão, que atualiza a posição dos vértices dos objetos a partir da utilização das equações da segunda Lei de Newton.

A idéia utilizada é a representação dos vértices dos objetos 3D como massas pontuais e as arestas são representadas por molas, cuja dinâmica obedece à Lei de Hooke [34]. Inicialmente, cada mola possui um comprimento padrão, que caso seja diminuído com o passar do tempo, impulsionará suas extremidades na direção oposta, e vice-versa. Neste modelo, cada elemento é atualizado pelas forças elásticas sendo aplicadas nas suas molas adjacentes.

Para eliminar oscilações infinitas nas molas, e simular a perda de energia que acontece no mundo real, adicionam-se amenizações às molas, conhecidas como *damping*. Existem vários modelos de aplicação de *damping*, como a adição de viscosidade artificial, *damping* linear ou *damping* de Rayleigh [12] [33].

Janson e Vergeest [35] propõem um modelo massa-mola que reduz a resolução (Figura 11) do modelo recursivamente, baseado em estimativas que garantem a sua corretude física. Seus autores classificam este modelo como um sistema massa-mola estendido, pois é baseado nos mesmos princípios destes, como a utilização de massas pontuais e a relação entre precisão e resolução do modelo.

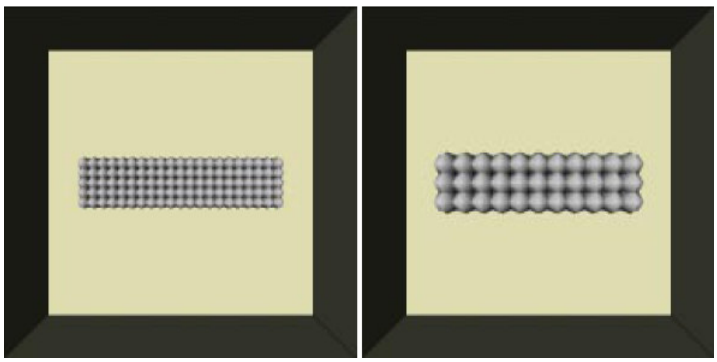

Figura 11. Modelos físicos 3D em resoluções diferentes.

Nesta técnica parte-se do pressuposto de que, se é possível afirmar que uma resolução ideal é fisicamente correta e a operação de redução desta resolução mantém esta propriedade, assume-se que resoluções mais baixas serão fisicamente

corretas. Tais modelos com resoluções diferentes são então mapeados de forma aproximada nos modelos gráficos, a fim de simular mais fielmente o comportamento.

A Figura 12 exibe alguns resultados desta técnica, ilustrando o mapeamento dos modelos 3D nos modelos físicos. O principal alvo de utilização idealizado pelos seus autores se enquadra em modelagem (v*irtual claying*) e prototipação rápida utilizando simulação em tempo real, a fim de estimar o comportamento de seus objetos rapidamente e com baixo custo.

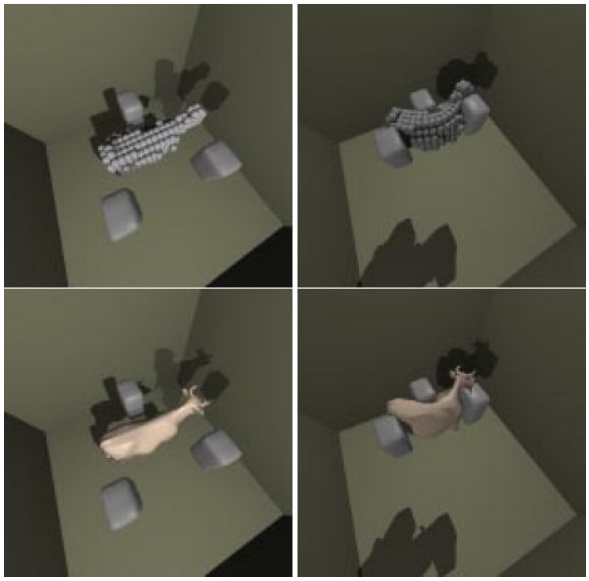

Figura 12. Modelos físicos mapeados nos modelos gráficos, como resultado de [35].

Como vem sendo notado, várias técnicas de tempo real utilizam modelos massa-mola pela sua simplicidade e facilidade de execução. As técnicas implementadas neste Trabalho de Graduação possuem fortes ligações com estes modelos, cada uma com características singulares [10] [12].

Todavia, segundo [12], molas não são os melhores modelos para representar tecidos ou objetos deformáveis reais, uma vez que seu alongamento é proporcional à força aplicada, o que pode gerar deformações inconsistentes.

Para contornar esta razoável inadequação do modelo massa-mola para simular deformação realista, existem correções de vários tipos para assegurarem que os objetos se comportarão de forma fisicamente plausível; estas correções são conhecidas como restrições (*constraints*) [36].

Várias técnicas utilizam esta abordagem para corrigir erros provenientes de aproximações, visto que cálculos contínuos ainda continuam sendo utilizados na simulação.

Em [37], por exemplo, é proposto um algoritmo de simulação de objetos volumétricos deformáveis, incluindo detecção de colisão somada a um passo posterior de relaxamento (*relaxation*) para minimizar o ganho desnecessário de energia ao final de cada passo da simulação. Para simular objetos volumétricos com mais exatidão, são criados elementos volumétricos (*voxels*) e cada elemento está conectado aos seus seis possíveis vizinhos no espaço. A Figura 13 ilustra em 2D como é a interação entre estes elementos.

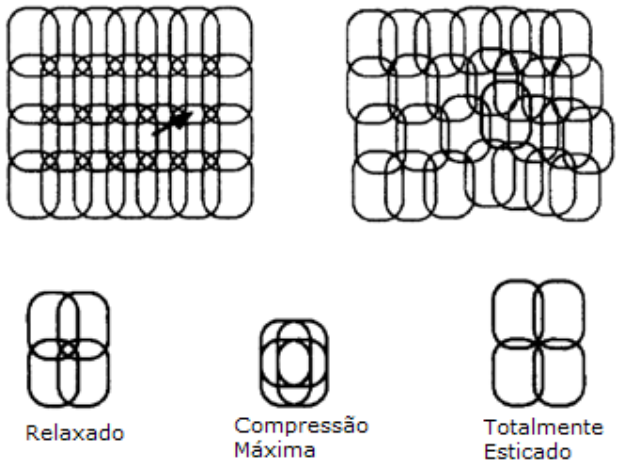

Figura 13. Modelo 2D do algoritmo proposto em [37] demonstrando o estado de seus elementos ao mover a porção apontada.

O seu principal foco de aplicação é a simulação cirúrgica. Segundo sua autora, poderia ser utilizado o FEM para cálculos em modelos de menor resolução, porém os *scanners* médicos produzem modelos de alta resolução, subutilizando o equipamento. Com este algoritmo, desta forma, é possível realizar cálculos com os modelos de resolução alta com taxas interativas.

Em [38] é proposta a utilização do Método de Elemento Longo (*Long Element Method* – LEM), que foi basicamente criado para simular objetos preenchidos com fluido [39]. Este trabalho faz parte de um simulador médico para exames em tecidos moles.

Trabalhos utilizando o FEM ou suas derivações também são capazes de alcançar taxas interativas de execução, desde que usados sob algumas restrições. Em

[17], é utilizada uma das derivações do FEM, o Método dos Elementos das Bordas (*Boundary Element Method* – BEM), que é capaz de executar em aplicações de tempo real pelo fato de que realiza os cálculos apenas da superfície do objeto, enquanto o FEM trata o volume completo.

Apesar da vantagem de poder executar em tempo real, segundo [40] e [12], este método utiliza modelos estáticos, tornando, desta forma, seu comportamento dinâmico menos realista.

Outro exemplo é a utilização de operadores diferenciais que, segundo [40], são fortemente relacionados ao formalismo matemático do FEM juntamente a conceitos do FVM. A deformação do corpo é feita localmente, ou seja, a matriz global do sistema não precisa ser totalmente resolvida, mas sim apenas a matriz relacionada ao nó sendo estimulado e seus vizinhos diretos.

A fim de incrementar ainda mais o desempenho de aplicações de simulação de objetos deformáveis em tempo real, além da utilização de modelos mais simplificados, como os vistos anteriormente, tem havido esforços no sentido de realizar a implementação deste tipo de simulação nos processadores gráficos (GPUs), aproveitando-se da sua altíssima capacidade de processamento e paralelização.

Ranzuglia *et al.* [41] propõem um método para simular elementos como correntes, malhas triangularizadas e malhas tetraédricas através de um sistema massa-mola totalmente executado na GPU, obtendo resultados razoavelmente satisfatórios – apenas o dobro do desempenho com relação à execução em um processador de uso geral – considerando que uma GPU como a NVIDIA GeForce 7800 GTX consegue cerca de 165 *gigaflops*, contra os 8 de um Pentium 4.

Segundo seus autores, este relativamente baixo ganho de desempenho é devido ao alto número de operações com dependências existentes durante a simulação, ou seja, operações que necessitam aguardar que outras sejam concluídas para que a simulação possa continuar.

Mosegaard e Sørensen [42] propõem uma separação no mapeamento entre o modelo de renderização gráfica e o modelo físico, uma vez que se esta relação das massas pontuais para com os vértices do elemento gráfico for um-para-um, a aparência pode ser comprometida.

A Figura 14 mostra a comparação entre um modelo mapeado numa relação de um-para-um com o modelo físico e a utilizando a forma de mapeamento proposta, deixando o objeto mais fielmente renderizado.

Figura 14. a) Modelo com mapeamento um-para-um entre o físico e o gráfico. b) Representação gráfica apenas do modelo físico (sistema massa-mola). c) Resultado esperado utilizando o mapeamento proposto por [42].

5. Estudos de caso: Técnicas desenvolvidas

Este capítulo detalha as técnicas de simulação física de objetos deformáveis implementadas neste Trabalho de Gaduação, a saber, *Position Based Dynamics* (PBD) [10] e de Desbrun *et al.* [12] enfatizando as suas principais características. Em seguida, detalhes da implementação de ambas as técnicas serão explicados, levando à constatação dos resultados obtidos, exibidos posteriormente. Finalizando, as principais dificuldades de execução do desenvolvimento serão descritas.

As técnicas PBD e a de Desbrun *et al.* foram escolhidas para a implementação primeiramente pelo fato de serem de simples execução e viáveis para aplicações de tempo real. A primeira utiliza o método de integração explícito de Euler e é orientada a restrições, o que garante uma melhor estabilidade à simulação. A técnica de Desbrun *et al.* utiliza uma aproximação de um modelo de integração implícito, com passos corretivos para estabilizar a simulação.

O fato de cada uma possuir um modelo de integração diferente permite que se obtenha um conhecimento mais abrangente dos principais conceitos que envolvem estes dois modelos.

Apesar de serem voltadas para a simulação de elementos deformáveis em geral, ambas as técnicas implementadas enfatizam a utilização de modelos planos, de forma a simular objetos neste formato, como tecidos, plantas, etc.

Baseado nisto, o principal foco dos casos de teste utilizados na implementação deste trabalho foi a utilização de malhas triangularizadas planas. Aliado a isto, também foram executados testes com malhas triangularizadas fechadas, com a intenção de avaliar a capacidade das técnicas na representação deste tipo de objeto 3D.

O grande problema com este tipo de malha para representar objetos volumétricos é a tendência deste tipo de objeto de tentar manter uma forma estável fisicamente, porém visualmente incoerente, devido ao fato de seu interior ser oco.

5.1. Técnica *Position Based Dynamics*

Considerada como um modelo mais generalizado da técnica proposta por Jakobsen [43], a técnica proposta por Müller *et al.* [10] é classificada como baseada em posição e restrições, pois a partir da resolução de restrições aplicadas aos vértices do sistema, as novas posições são aplicadas sem inconsistências. Somado a isto, a manipulação direta das posições dos vértices proporciona naturalmente um maior

controle e estabilidade ao sistema. A seguir, uma visão geral do algoritmo é apresentada.

5.1.1. O sistema físico

Considerando um objeto 3D composto de N vértices e M restrições, tem-se que cada vértice i pertencente ao objeto possui uma posição x_i, uma massa m_i e uma velocidade v_i.

Além disto, cada restrição j é composta por:

- Uma cardinalidade n_j, que indica sobre quantos vértices ela atua;
- Uma função $C_j: \mathbb{R}^{3n_j} \longrightarrow \mathbb{R}$, cujo domínio são as coordenadas das partículas sobre as quais atua, cada uma com três graus de liberdade, resultando num valor real que determinará se a restrição deverá ser aplicada;
- Um conjunto de índices, relativos aos vértices do objeto 3D, cuja cardinalidade é igual a n_j;
- Um parâmetro de dureza (*stiffness*) k, que assume valores reais entre 0 e 1 e define o quão atuante é a restrição sobre seus vértices relacionados;
- Um tipo que especifica se a restrição deverá ser satisfeita quando $C_j\left(x_{i_1}, \ldots, x_{i_{n_j}}\right) = 0$ (*equality constraint*) ou se é satisfeita quando $C_j\left(x_{i_1}, \ldots, x_{i_{n_j}}\right) \leq 0$ (*inequality constraint*).

Com estas informações iniciais e considerando-se uma variação de tempo (*timestep*) Δt, o algoritmo apresentado na Figura 15 se comportará da maneira detalhada nas próximas subseções.

```
(1)  forall vertices i
(2)      initialize x_i = x_i^0, v_i = v_i^0, w_i = 1/m_i
(3)  endfor
(4)  loop
(5)      forall vertices i do v_i ← v_i + Δt w_i f_ext(x_i)
(6)      dampVelocities(v_1, ..., v_N)
(7)      forall vertices i do p_i ← x_i + Δt v_i
(8)      forall vertices i do generateCollisionConstraints(x_i → p_i)
(9)      loop solverIterations times
(10)         projectConstraints(C_1, ..., C_{M+M_coll}, p_1, ..., p_N)
(11)     endloop
(12)     forall vertices i
(13)         v_i ← (p_i − x_i)/Δt
(14)         x_i ← p_i
(15)     endfor
(16)     velocityUpdate(v_1, ..., v_N)
(17) endloop
```

Figura 15. O algoritmo em pseudocódigo do PBD [10]

5.1.2. Inicialização e predições iniciais

Nas linhas 1 até 3 do algoritmo mostrado na Figura 15, são inicializados todos os vértices e sua posição inicial, velocidade inicial e massa, além de uma variável w, que é o inverso da massa, utilizada principalmente na projeção das restrições, posteriormente.

No *loop* principal do algoritmo atualiza-se a velocidade de cada vértice considerando a atuação das forças externas ao sistema, quando não se podem utilizar restrições de posição para representá-las. No caso deste algoritmo, apenas a força da gravidade é adicionada ao sistema neste passo (linha 5).

Na linha 6, as velocidades dos vértices são amenizadas (*damping*). Isto é feito através do cálculo da variação da velocidade do vértice, subtraindo sua velocidade individual da movimentação global do objeto. Depois disso, a velocidade individual é calculada utilizando-se uma constante k, que assume valores reais entre 0 e 1, como um peso para controlar a contenção na velocidade individual. Desta forma, se o valor de k for igual a 1, a velocidade de cada vértice em relação ao centro de massa do objeto não variará, resultando no comportamento semelhante ao de um corpo rígido [10].

No próximo passo (linha 7), são calculadas as novas posições de cada vértice utilizando uma integração Euleriana explícita ($x = x_0 + v\Delta t$), baseando-se na posição atual e na sua velocidade recém calculada.

5.1.3. Restrições

Neste algoritmo são propostas duas classificações para as restrições a serem manipuladas: as fixas e as dinâmicas. As fixas são as que sempre terão de ser satisfeitas, não variando ao longo da simulação, ou seja, elas poderão ser criadas antes da execução do algoritmo. As dinâmicas são as que precisam ser geradas a cada iteração, pois suas condições variam com o tempo.

As restrições fixas podem ser pré-computadas antes da inicialização, sendo necessário apenas a sua projeção durante as iterações do solver (linha 9). Alguns exemplos deste tipo de restrições são:

- Restrição de distância: mantém os vértices do objeto a uma distância determinada;

- Restrição de pressão interna: utilizado em malhas fechadas para simular objetos semelhantes a balões;
- Restrição de auto-colisão: aplica colisão dos vértices da malha aos outros vértices da mesma malha.

O principal exemplo de restrições dinâmicas são as de colisão, que tem tanto suas condições, quanto seus vértices relacionados sendo modificados a cada iteração.

Na linha 8 do algoritmo (Figura 15), as novas restrições de colisão da iteração são geradas.

Para a geração de restrições de colisão utilizam-se duas abordagens a fim de garantir que não haverá falha na colisão. Primeiramente, um esquema denominado de colisão contínua é aplicado, da seguinte maneira: traça-se um raio da posição anterior (representada por x_i) à posição prevista no passo da linha 7 do algoritmo, p_i.

Se o raio intersectar a superfície de um objeto, calcula-se o ponto de entrada do raio q_c e a normal da superfície naquela posição, n_c, resultando numa nova restrição de desigualdade $C(p) = (p - q_c) \cdot n_c$, com $k = 1$.

Esta restrição é de desigualdade por que, evidentemente, a nova posição deverá estar fora do corpo colidido para satisfazer à restrição, ou seja, o cosseno do ângulo entre os vetores $< p - q_c >$ e n_c deverá ser menor ou igual a zero.

Por outro lado, se o raio $x_i \longrightarrow p_i$ entra totalmente no objeto, o método descrito anteriormente falhou em algum ponto, pois até a posição anterior do vértice já está dentro do objeto colidido. Neste caso, a saída é utilizar um método de colisão estático, calculando-se o ponto mais próximo de p_i na superfície, q_s, e a normal naquela posição da superfície n_s, dando origem a uma nova restrição de desigualdade $C(p) = (p - q_s) \cdot n_s$, também com $k = 1$.

A utilização do coeficiente de *stiffness* (k) nas restrições, a fim de potencializar ou amenizar sua atuação, é feita da seguinte forma: segundo [10], há vários métodos de se incorporar o *stiffness* às restrições, sendo o mais simples a multiplicação do valor de k pelas correções encontradas no final das projeções. Porém, o efeito desta multiplicação vai perdendo sua linearidade com o aumento no número de iterações do *solver*.

Desta forma, para que a relação seja sempre linear, a multiplicação deverá ser feita por $k' = 1 - (1 - k)^{1/n_s}$, em que n_s é o número de iterações do *solver*.

As restrições de igualdade sempre terão de ser projetadas visto que o resultado de sua função deverá ser unicamente igual a zero. As de desigualdade serão projetadas sempre que o resultado da avaliação de sua função for maior que zero como no caso da restrição de colisão, por exemplo.

5.1.4. Solver

O próximo passo é o *solver* do algoritmo, que é o seu *loop* mais importante, pois é nele que são verificadas e projetadas todas as restrições (linhas 9 a 11). Ele consiste em um laço com um determinado número de iterações que executa uma função que verifica e projeta todas as restrições, tanto as fixas quanto as de colisão recém-geradas, atualizando as próximas posições dos vértices.

Projetar os pontos de acordo com uma determinada restrição é movê-los de forma que a restrição seja satisfeita. A conservação dos momentos linear e angular cria as duas principais restrições internas a serem satisfeitas durante a projeção, para que se tenha uma simulação estável, isto é, sem forças desconhecidas gerando translações ou rotações indesejadas no objeto, conhecidas como *ghost forces* [10].

O método apresentado para a projeção das restrições funciona de tal forma que as restrições de momento são satisfeitas automaticamente, de uma forma geral. Isto é possível pelo fato de que, utilizando os p_n pontos sobre os quais uma restrição C qualquer atua e concatenando-os na forma de $p = [p_1^T, \ldots, p_n^T]^T$, para restrições internas, a translação e a rotação do objeto não afetarão o valor de C. Dessa forma, $\nabla_{\mathrm{p}} C$, é perpendicular tanto à rotação quanto à translação, dado que 90º é a direção de variação máxima. Por isso, se a variação da posição Δp ocorre ao longo $\nabla_{\mathrm{p}} C$, os momentos são conservados.

Para encontrar Δp dado que se possui p, tal que $C(p + \Delta p) = 0$, tem-se que:

$$C(p + dp) \approx C(p) + \nabla_{\mathrm{p}} C(p) \cdot \Delta p = 0. \tag{9}$$

E, sabendo-se que para que Δp tenha a mesma direção que $\nabla_{\mathrm{p}} C$, deve-se ter um λ tal que:

$$\Delta p = \lambda \nabla_{\mathrm{p}} C(p) \tag{10}$$

Substituindo-se Δp na equação (9) pela equação (10) e isolando-se λ, tem-se:

$$\Delta p = -\frac{C(p)}{\left|\nabla_p C(p)\right|^2}\nabla_p C(p). \quad (11)$$

Chegando assim na equação para a correção da posição de um ponto individual:

$$\Delta p_i = -s\,\nabla_{p_i} C(p_1,\ldots,p_n)\,, \quad (12)$$

em que s é:

$$s = \frac{C(p_1,\ \ldots,p_n)}{\sum_j |\nabla_{p_j} C(p_1,\ldots,p_n)|^2}. \quad (13)$$

Caso haja diferença nas massas dos vértices, pesam-se as correções Δp_i, com o valor inverso da massa w_i. Assim:

$$\Delta p_i = \lambda w_i \nabla_{p_i} C(p)\,, \quad (14)$$

resultando em:

$$\nabla p_i = -s w_i \nabla_{p_i} C(p_1,\ldots,p_n)\,, \quad (15)$$

dado que

$$s = \frac{C(p_1,\ldots,p_n)}{\sum_j w_j |\nabla_{p_j} C(p_1,\ldots,p_n)|^2}. \quad (16)$$

5.1.5. Atualizações

Depois das projeções, tendo as novas posições dos vértices respeitando todas as restrições consideradas, pode-se iterar por todos os vértices atualizando as velocidades v_i e posições x_i de cada um (linhas 13 e 14).

O último passo do algoritmo (linha 16) atualiza as velocidades para todos os vértices que tiveram restrições de colisão geradas, refletindo sua posição perpendicularmente à superfície colidida, aplicando restituição e atrito, caso necessário.

5.2. Técnica de Desbrun *et al.*

O algoritmo proposto por Desbrun *et al.* [12] usa um modelo de animação de sistemas massa-mola utilizando uma aproximação da idéia de integração implícita aliado a um passo posterior de correção das estimativas para conservar o momento.

Juntamente a isso, utiliza um processo que utiliza cinemática reversa para manipular possíveis colisões e satisfazer certas restrições, como manter uma distância aceitável entre vértices.

De uma forma geral, a aproximação proposta pretende realizar a atualização das posições dos vértices (massas pontuais) com a facilidade de um modelo explícito Euleriano unido às propriedades principais de um esquema implícito, dentre as quais se pode citar uma maior estabilidade do sistema, por exemplo [12]. A Figura 16 ilustra através de pseudo-código o algoritmo em questão.

Precompute $W = (\mathbf{I}_n - \frac{dt^2}{m}H)^{-1}$
At each time step dt
  //Compute internal forces $\mathbf{F}_i$
  //due to springs and artificial viscosity.
  $\mathbf{x}_G = \mathbf{0}$
  For each mass point i
    $\mathbf{F}_i = \mathbf{0}$
    $\mathbf{x}_G += \mathbf{x}_i$
    For each mass point j such as (i,j) linked by a spring
      $\mathbf{F}_i += k_{ij}\,(||\mathbf{x}_i - \mathbf{x}_j|| - l_0^{ij})\,\frac{\mathbf{x}_i - \mathbf{x}_j}{||\mathbf{x}_i - \mathbf{x}_j||}$
      $\mathbf{F}_i += k_{ij}\,dt\,(\mathbf{v}_j - \mathbf{v}_i)$
  $\mathbf{x}_G /= n$
  $\delta T = \mathbf{0}$
  // Integrate the approximation (predictor)
  For each mass point i
    $\mathbf{F}_i^{filtered} = \sum_j \mathbf{F}_j W_{ij}$
    $\delta T += \mathbf{F}_i^{filtered} \wedge \mathbf{x}_i$
    $\mathbf{v}_i^{n+1} = \mathbf{v}_i^n + \left[\mathbf{F}_i^{filtered} + \mathbf{F}_i^{ext}\right]\frac{dt}{m}$
    $\mathbf{x}_i^{new} = \mathbf{x}_i + \mathbf{v}_i^{n+1}\,dt$
  // Post correction of angular momentum (corrector)
  For each mass point i
    $\mathbf{R}_i^{correc} = (\mathbf{x}_G - \mathbf{x}_i) \wedge \delta T$
    $\mathbf{x}_i^{new} += \mathbf{R}_i^{correc}\frac{dt^2}{m}$
  // Now, use the inverse kinematics
  $nbIter = 0$
  do
    Post-step inverse kinematics
    $nbIter = nbIter + 1$
  until $(error < \varepsilon)$ or $(nbIter > nbIterMax)$ or *(time is up!)*
  // Update real velocity and position
  $\mathbf{v}_i^{n+1} = (\mathbf{x}_i^{n+1} - \mathbf{x}_i^n)/dt$
  $\mathbf{x}_i = \mathbf{x}_i^{new}$

Figura 16. Pseudo-código do algoritmo proposto por Desbrun *et al.* [12].

5.2.1. Modelo simplificado

Para um entendimento dos fundamentos desta técnica, primeiramente é analisado um modelo de uma dimensão. Desta forma, é considerado um sistema massa-mola em que cada vértice de massa m_i e posição x_i está ligado a no máximo dois vizinhos através de uma representação de mola com constante de *stiffness* k.

Considerando um sistema físico com estas características iniciais, tem-se:

- x representará as posições dos vértices do sistema, no caso das x_i massas pontuais $x = (x_1, x_2, \dots, x_n)^T$;

- v é a representação das velocidades v_i do sistema, tal que $v = \dot{x}$;
- F_i representa as forças internas (relativas às molas) agindo sobre uma massa i, em que F_i^{ext} representa as forças externas, como a gravidade, por exemplo;
- Elementos sobrescritos indicam um tempo iniciado por um tempo arbitrário t_0. Por exemplo, $x_i^n = x_i(t_0 + n\,dt)$;
- O operador de diferença também será utilizado: $\Delta^{n+1}x = x^{n+1} - x^n$.

Para simular o sistema, vários métodos de integração podem ser utilizados, como a integração Euleriana explícita, por exemplo.

Devido a suas desvantagens, como descritas no capítulo 2, pode-se utilizar como alternativa a este método a integração implícita Euleriana que, segundo [44] se mostra mais adequada pelas suas vantagens com relação à estabilidade. A diferença entre a integração implícita e a explícita é a substituição das forças no tempo t por forças no tempo $t + dt$, como em:

$$v_i^{n+1} = v_i^n + F_i^{n+1}\frac{dt}{m} \tag{17}$$

$$x_i^{n+1} = x_i^n + v_i^{n+1}dt \tag{18}$$

Diferentemente das equações (7) e (8) (capítulo 2), neste esquema as novas posições não são inadvertidamente encontradas, mas somente quando o campo de forças estiver numa configuração coerente. Assim, teoricamente, em qualquer valor de *timestep* as forças geradas serão consistentes, sem gerar instabilidades [8].

Com esses dados, deve-se computar F^{n+1} sem conhecer ainda as posições das massas no tempo $t + dt$. Desta forma, pode-se construir uma aproximação de primeira ordem que, para molas é exata [12]:

$$F^{n+1} = F^n + \frac{\partial F}{\partial x}\,\Delta^{n+1}x. \tag{19}$$

$H = \frac{\partial F}{\partial x}$ é a matriz Hessiana negada do sistema que, no caso de uma dimensão, é a seguinte:

$$H = k \begin{bmatrix} -1 & 1 & 0 & 0 & . & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & . & 0 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & . & 0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & 0 & . & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 0 & . & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & . & 0 & 0 & 1 & -1 \end{bmatrix}. \quad (20)$$

Substituindo-se a equação (19) na equação (17) e sabendo-se que $\Delta^{n+1}x = (v^n + \Delta^{n+1}v)dt$, chega-se a:

$$\Delta^{n+1}v = (\mathbf{I} - \frac{dt^2}{m}H)^{-1}(F^n + dt\,Hv^n)\frac{dt}{m}. \quad (21)$$

$\tilde{F}^n = dt\,Hv^n$ são forças adicionais dissipativas que são adicionadas ao conjunto das forças internas F^n. Para um dado ponto i, a força adicional naquele ponto $\tilde{F}_i$ pode ser escrita da seguinte maneira, graças à estrutura de H:

$$\tilde{F}_i = k\,dt \sum_{j|(i,j)\in Arestas} (v_j - v_i)\ . \quad (22)$$

Ou seja, a soma das velocidades relativas entre vértices vizinhos é utilizada como viscosidade em modelos massa-mola ou sistemas de partículas, o que significa que o movimento de cada ponto será influenciado pela movimentação de seus vizinhos.

Na integração implícita, este tipo de viscosidade é proporcional ao *timestep* e ao *stiffness* do material. Essas variáveis são bastante conhecidas como geradoras de instabilidade no processo de integração, e por isso as forças são amenizadas utilizando-se uma determinada quantidade de viscosidade, para adicionar estabilidade.

Depois de adicionada a viscosidade ao sistema, a variação da velocidade é calculada através da multiplicação das forças resultantes pela seguinte matriz:

$$W = (\mathbf{I} - \frac{dt^2}{m}H)^{-1}\ . \quad (23)$$

A multiplicação do campo de forças por *W* é equivalente a uma filtragem, ou seja, as forças resultantes são uma convolução discreta entre as forças e este conjunto de filtros contidos em *W* [12].

5.2.2. Modelos 2D e 3D

Generalizando para modelos de objetos mais complexos, como superfícies e objetos volumétricos, são utilizadas massas pontuais i , que são ligadas por representações de molas com comprimento inicial (ou de repouso) l_{ij}^{0} e coeficiente de *stiffness* k_{ij}.

Visto que o modelo possui suas principais características derivadas de modelos implícitos, é necessário primeiramente calcular a matriz Hessiana do sistema, porém diferentemente do modelo unidimensional descrito anteriormente, em modelos mais complexos, a matriz Hessiana do sistema não é mais constante, tornando-se necessário o cálculo de uma matriz $3\mathrm{n} \times 3\mathrm{n}$ com somas de expressões como a seguinte [12]:

$$\frac{\partial F_{(i,j)}}{\partial x_i} = -k_{ij}\left[\frac{\|x_i - x_j\| - l_{ij}^{0}}{\|x_i - x_j\|}\mathrm{I}_3 + l_{ij}^{0}\frac{(x_i - x_j)^T(x_i - x_j)}{\|x_i - x_j\|^3}\right]. \tag{24}$$

Resolvendo o sistema linear resultante para cada *timestep* certamente se tornará muito custoso para execução em tempo real. Desta forma, uma abordagem alternativa foi proposta por Desbrun *et al.*, que elimina a necessidade de resolver um sistema linear tão complexo, executando uma aproximação da matriz.

Para realizar a aproximação, o cálculo das forças internas resultantes é simplificado através da separação destas forças em lineares e não lineares. Assim:

$$F_{(i,j)} = F_{(i,j)}^{linear} + F_{(i,j)}^{\sim linear}, \tag{25}$$

em que as forças não lineares são as que fazem a mola que une dois vértices apenas rotacionar. Por isso, segundo [12], elas possuem a característica de não variar a sua magnitude. Então, considerando que esta característica adicionará apenas um erro de ângulo no sistema, estas forças serão simplesmente deixadas de lado, sendo aplicado um passo de correção para estes erros gerados.

As forças lineares agem sobre os vértices independentemente do comprimento de repouso da mola. Este tipo de força é o mesmo modelo mais simples discutido na seção anterior, conseqüentemente, sua matriz Hessiana é constante, simplificando sua criação para a seguinte expressão:

$$H = \begin{cases} H_{ij} = k_{ij}, & se\ i \neq j \\ H_{ii} = -\sum_{j \neq i} k_{ij} \end{cases} . \quad (26)$$

Esta matriz pode ser gerada apenas uma vez para ser utilizada durante toda a simulação, independentemente do *timestep*, utilizando a equação (21).

5.2.3. Conservação dos momentos

Diferentemente da técnica PBD [10], que conserva naturalmente tanto o momento angular quanto o linear, esta técnica conserva apenas o momento linear, visto que o somatório das forças de viscosidade é igual a zero [12]:

$$\sum_{i=1}^{n} \tilde{F}_i = dt \sum_{i=1}^{n} \left(\sum_{j=1}^{n} k_{ij}(v_j - v_i) \right) = dt \sum_{i<j} k_{ij}\left[(v_j - v_i) + (v_i - v_j)\right] = 0 \quad (27)$$

O momento angular não é preservado, como explicado na subseção anterior. No entanto, há um método de correção, que é executado da seguinte maneira: depois das forças internas estarem filtradas, ou seja, multiplicadas por W, a matriz constante, calcula-se o torque global no objeto δT utilizando a equação

$$\delta T = \sum_{i=1}^{n} F_i^{filtrada} \times x_i . \quad (28)$$

Este valor de torque deverá ser nulo, a fim de garantir a conservação do momento, então se adiciona às novas posições dos vértices o seguinte vetor de correção:

$$R^{correção} = (x_G - x_i) \times \delta T \frac{dt^2}{m} \quad (29)$$

em que x_G é o centro de gravidade do corpo. Vale ressaltar que a soma de todos os vetores de correção é igual a zero, modificando – e corrigindo – apenas o momento angular, não afetando o momento linear.

Segundo [12], esta abordagem, por compensar o erro no torque global, poderá causar algumas variações locais de torque, porém este tipo de problema só se torna aparente, de forma a atrapalhar a simulação, para constantes de *stiffness* muito altas ou *timesteps* longos.

5.2.4. Passo posterior

A fim de evitar problemas relativos à elongação muito grande das molas do sistema, se faz necessária uma modificação em seu comportamento para torná-lo mais realista, visto que a elongação de molas com relação à força aplicada não possui natureza linear.

Este passo posterior pode ser considerado como um reforço por restrição, visto que as posições dos vértices são deslocadas depois de se movimentarem normalmente, após cada iteração.

Existem vários métodos para executar este reforço, como em [45] que Provot utiliza uma abordagem que caso a mola esteja esticada demais, os dois vértices ligados por ela são trazidos de volta seguindo seus eixos, sempre conservando o centro de gravidade de ambas as massas pontuais. Caso um desses pontos esteja preso a uma posição, deve-se mover apenas o seu vizinho, até atingir uma elongação plausível.

Por se tratar de cinemática reversa, sem o envolvimento de forças, não há problemas com relação à estabilidade ou conservação de momentos.

5.3. Implementação do protótipo em tempo real

Ambas as técnicas possuem como principal característica a simplicidade de execução. Por este motivo, e a fim de manter o foco exclusivamente na resolução de problemas relacionados a corpos deformáveis, optou-se por utilizar, numa primeira iteração do trabalho, o motor gráfico *open source* OGRE 3D [46] para a renderização gráfica da cena. Desta forma, a implementação de estruturas e operações relacionadas tanto ao contexto gráfico quanto ao físico pôde ser abstraída.

Percebe-se que, devido ao nível de abstração inerente aos motores gráficos, haverá certamente diminuição no desempenho da aplicação em algum nível. Porém, nos testes que foram realizados, o prejuízo quanto a este fator não foi tão notável. Restrições quanto ao escopo e tempo do desenvolvimento do Trabalho de Graduação também influenciaram esta escolha.

5.3.1. Estrutura geral

Ambas as técnicas desenvolvidas puderam ser estruturadas de forma similar, principalmente devido ao modelo de programação do OGRE. Em linhas gerais, a estrutura de ambas a implementações pode ser descrita da seguinte maneira:

- Uma classe de tratamento dos modelos gráficos, que é por onde é inicializada a aplicação;
- Uma classe para a manipulação dos cálculos físicos (*solver*), que recebe os elementos necessários para a execução, como posições iniciais dos vértices, normais e informações de arestas;
- Classes básicas que representam as massas pontuais, molas, etc.;
- Uma classe de manipulação de eventos de interação e atualização da cena.

A classe de inicialização e tratamento de modelos gráficos distribui os dados iniciais para a classe de execução dos cálculos físicos e para a de tratamento de eventos. Desta forma, sempre que o *solver* atualizar as posições dos vértices, o sistema será atualizado. A diferença principal entre as duas implementações estão localizadas nas classes básicas e na implementação do solver de cada técnica. A Figura 17 ilustra este processo.

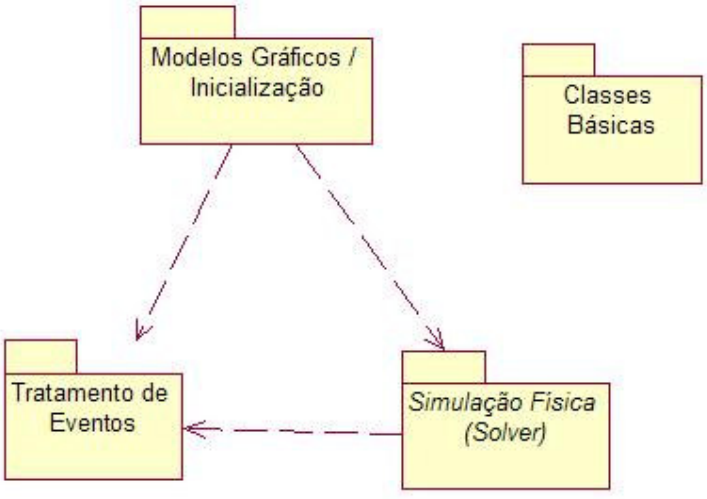

Figura 17. Modelo estrutural comum a ambas as implementações das técnicas.

Na classe que manipula os modelos gráficos, foi desenvolvido um pequeno leitor de arquivos do tipo WAVEFRONT OBJ, tornando mais fácil a aquisição de modelos a partir de ferramentas de edição gráfica, como o 3D Studio Max [21] ou o Maya [22], por exemplo.

5.3.2. Técnica *Position Based Dynamics*

A técnica proposta por Müller *et al.* possui como classe básica a representação de vértices (massas pontuais) `Vertex`, que contém propriedades como a posição inicial (x_i), a posição prevista (p_i), a massa (m_i), o inverso da massa (w_i) e a velocidade (v_i).

Além disso, a fim de representar as restrições, existe uma classe básica abstrata `Constraint`, da qual todas as restrições implementam seu método `project()` e herdam os atributos comuns a todas as restrições, como o *stiffness* e o seu tipo (igualdade ou desigualdade). O método `project()` é o responsável por projetar a restrição em questão durante a execução do *solver*.

5.3.3. Técnica de *Desbrun et al.*

A técnica proposta por Desbrun *et al.* possui a representação básica de massas pontuais, `Vertex`, englobando suas principais propriedades, como massa (m_i), posição (x_i), velocidade (v_i) e força aplicada (F_i). Além disso, há uma representação das molas que interconectam os vértices, na classe `Spring`. Cada mola possui uma referência a cada um dos vértices que conecta, além do valor de sua elongação de repouso (l_{ij}^0) e sua constante de *stiffness* (k).

As molas são criadas a partir da informação das arestas o objeto gráfico. Além destas molas, de forma a se criar uma simulação mais estável, se faz necessário criar as conhecidas molas de sustentação (*shear springs*) [45], que evitam que os quadrados da malha se deformem lateralmente. A Figura 18 ilustra um esquema que representa o modelo massa-mola utilizado para as malhas testadas. As molas em diagonal representam as *shear springs*.

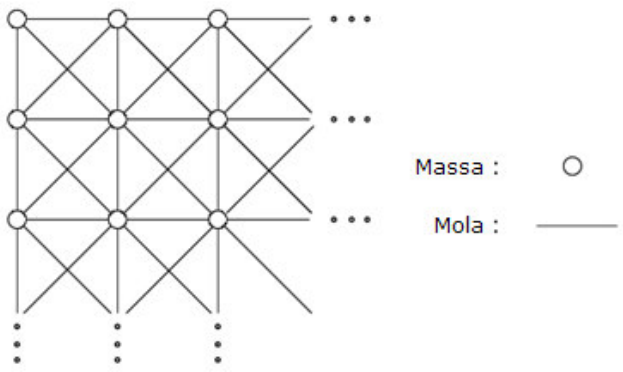

Figura 18. Modelo massa mola utilizado na técnica de Desbrun et al [12].

5.4. Resultados

Para as duas técnicas, o principal teste executado foi o de uma malha triangularizada suspensa por alguns vértices, que foram mantidos fixos. A máquina utilizada para os testes foi uma com processador *dual-core* Intel Core2Duo 2.2GHz, 2GB de memória RAM e uma placa gráfica NVIDIA Geforce série 8.

A técnica PBD se mostrou bem eficiente, considerando apenas uma restrição sendo executada por vez. Das restrições mostradas em [10], com apenas a restrição de distância já se consegue obter resultados que demonstram efeitos físicos razoáveis. Apesar desta técnica não fazer uso explícito do conceito de massa-mola como na técnica de Desbrun *et al.*, para a restrição de distância é necessária a utilização de conceitos semelhantes à utilização de molas. A Figura 19 mostra uma malha de 289 vértices com as *shear springs* e a restrição de distância aplicada.

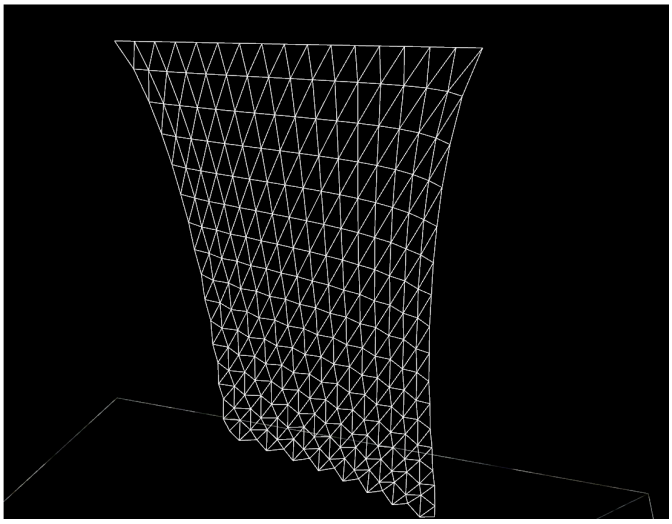

Figura 19. Malha sob o efeito da restrição de distância e com *shear springs*.

Os vértices superiores estão fixos. Nota-se que, quanto mais próximo das extremidades livres do objeto, mais ele vai ficando enrugado, este comportamento posa como uma das dificuldades encontradas. Suspeita-se que este efeito se deve ao fato de a restrição de distância possuir um valor alvo fixo, tornando o comportamento de algumas regiões do objeto similar ao de uma malha metálica (*chain-mail*).

Para demonstrar o efeito produzido pela adição de *shear springs*, a Figura 20 mostra o comportamento da mesma malha da figura anterior sem as molas especiais. Nela, pode-se notar que alguns quadrados da malha ficaram degenerados, gerando uma leve rotação do objeto.

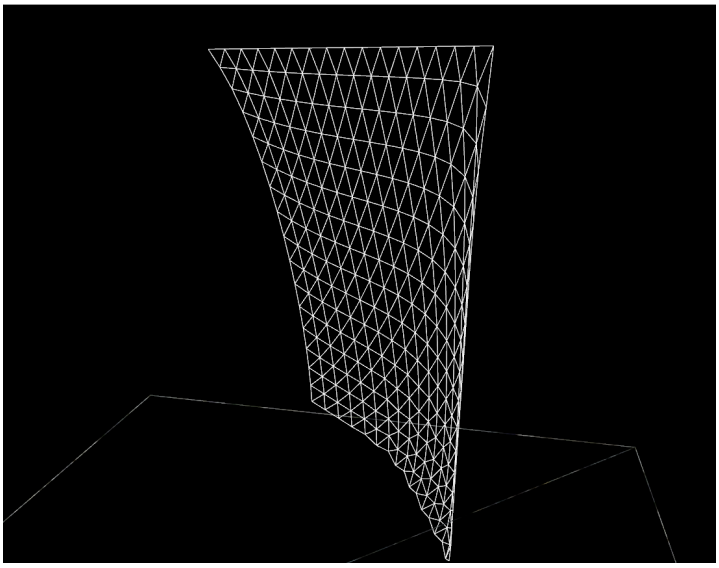

Figura 20. Malha sob o efeito da restrição de distância sem *shear springs*.

Vale ressaltar que, apesar da informação das arestas do objeto gráfico servir de base para a criação de molas, elas não são representadas graficamente, mas apenas as arestas dos triângulos dos objetos.

Além da restrição de distância, a restrição de pressão interna para malhas triangularizadas fechadas também foi implementada. Porém, infelizmente, não foi possível atingir os resultados esperados do ponto de vista gráfico, conforme exibido em [10], apesar do comportamento do corpo, depois de submetido à pressão da restrição, aparentar ser fisicamente satisfatório, ou seja, a cada aumento de pressão sofrido pelo objeto, sua densidade diminui, fazendo com que ele flutue de forma similar a um balão.

O principal problema com relação ao comportamento gráfico do objeto foi que a sua forma não consegue ser mantida adequadamente ao longo da simulação.

Para simular objetos volumétricos com malhas triangularizadas foi utilizado um modelo de aproximadamente 2500 vértices. Utilizando a restrição de distância e ajustando alguns parâmetros como o *stiffness*, torna-se capaz de fornecer uma visualização aproximada deste tipo de objetos.

A Figura 21 mostra uma malha fechada utilizando a restrição de posição. A do lado esquerdo mostra a malha com valor de *stiffness* igual a 1. A da direita tem valor de *stiffness* igual a 0.3. Baseado nisto, conclui-se que a utilização de modelos tetraédricos seria o mais adequado para uma representação mais eficiente e adequada.

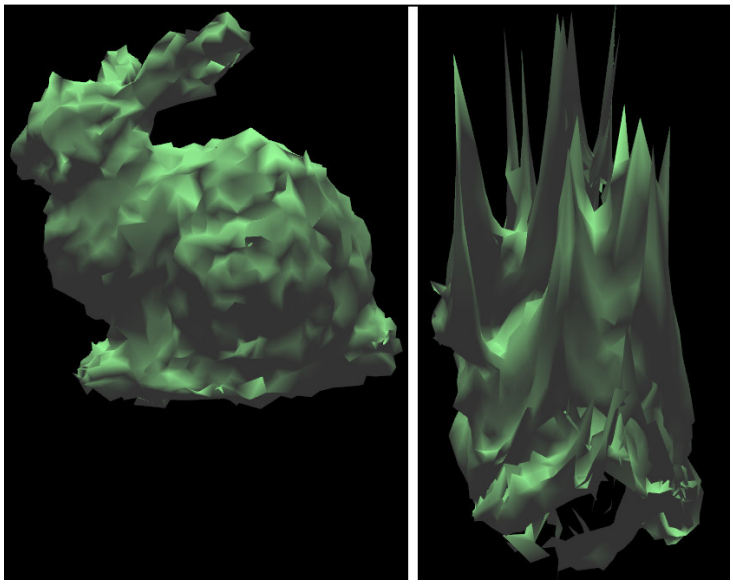

Figura 21. Comparação entre o formato de uma malha fechada com valores de *stiffness* diferentes.

A técnica de Desbrun *et al.* foi implementada completamente. Em relação ao PBD, esta obteve resultados mais satisfatórios do ponto de vista do realismo, apesar de a técnica PBD não ter tido todas as suas restrições implementadas [10].

A Figura 22 abaixo mostra um dos resultados (em *wireframe*) utilizando esta técnica para a mesma malha exibida anteriormente.

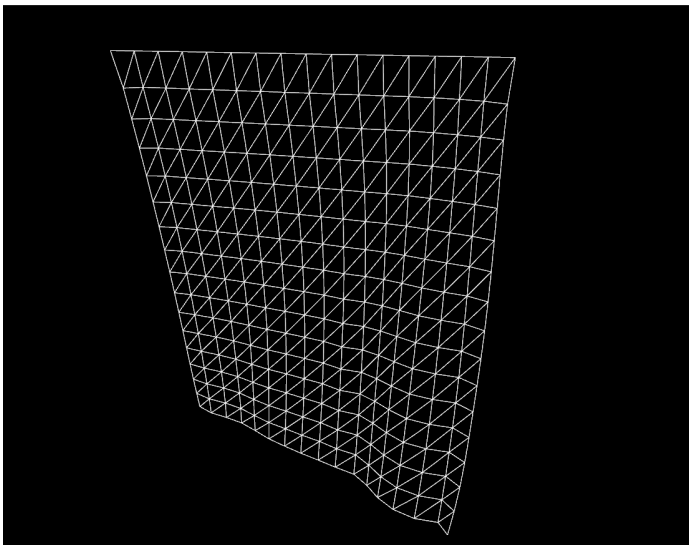

Figura 22. Malha com 289 vértices sendo simulada com a técnica de Desbrun *et al*.

A grande desvantagem desta técnica é o passo de pré-processamento muito custoso, necessário antes da execução da simulação. Este custo foi confirmado para malhas com mais de 500 vértices, assim como descrito em [12].

Para elementos com um número de vértices entre o valor proposto por Desbrun *et al.* (120 e 400), as taxas de quadros por segundo foram semelhantes às alcançadas no artigo que baseou parte deste trabalho de graduação. Por exemplo, a malha exibida na Figura 23, com 289 vértices chega a taxas de 50 quadros por segundo, mesmo com todo o custo adicional imposto pela utilização de um motor gráfico de alto nível (o OGRE).

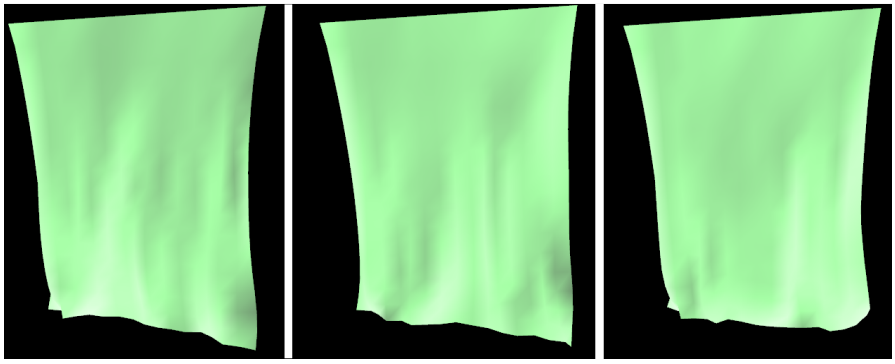

Figura 23. Seqüência de imagens da simulação da técnica de Desbrun *et al.*

Em termos comparativos, a técnica PBD possui uma capacidade de execução em tempo real mais eficiente que a técnica de Desbrun *et al.*, visto que esta possui o passo de pré-processamento e sua execução pode ser comprometida dependendo do tamanho ou resolução do objeto. O PBD consegue simular um objeto de mais de 2000 vértices a taxas médias de 50 quadros por segundo, utilizando as restrições implementadas.

5.5. Dificuldades encontradas

Grande parte das dificuldades encontradas ao longo do desenvolvimento deste trabalho está relacionada aos conceitos matemáticos que dão suporte a este tipo de simulação. Estas dificuldades dizem respeito tanto ao entendimento de tais conceitos como em encontrar formas computacionalmente eficientes para implementar certas deduções matemáticas. No PBD, estes conceitos podem ser considerados como os presentes na geração das restrições. Na técnica de Desbrun *et al.*, o cálculo das forças foi uma das maiores dificuldades encontradas.

Os resultados obtidos nas restrições implementadas para o PBD ficaram aquém do esperado, visto que não foi possível identificar as causas de incoerências, i.e.,

deformidades indesejadas nas malhas, como ilustrado na Figura 19 e na Figura 24. Apesar de seus comportamentos físicos serem plausíveis, como já mencionado.

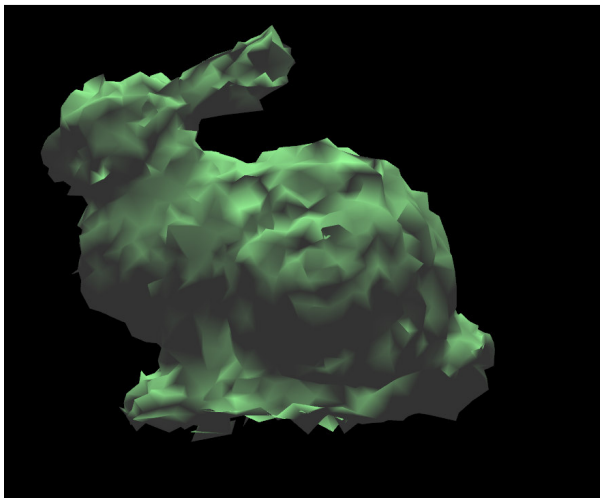

Figura 24. Malha utilizando a restrição de distância do PBD apresentando algumas inconsistências.

6. Conclusão

Aplicações de simulação física interativas necessitam de baixas latências, a fim de satisfazer seus usuários, provendo um tempo de resposta visualmente aceitável. Além disso, a interatividade é apenas uma parcela do processo de simulação, visto que o comportamento físico dos objetos deve possuir um nível plausível de realismo, de forma a convencer o usuário de que um determinado elemento virtual está se comportando como o seu modelo real. Este é o principal objetivo das técnicas de tempo real detalhadas neste Trabalho de Graduação.

Técnicas baseadas em posição têm como principal característica a eficiência, devido à sua relativa simplicidade computacional e adequação visual. A tendência é que, quanto mais restrições o sistema físico possuir, no caso do PBD, a complexidade matemática cresça, melhorando a precisão e possivelmente reduzindo a velocidade de execução.

Técnicas que utilizam modelos de integração implícitos são mais custosas, devido ao fato de ser necessária a geração de sistemas lineares demasiadamente extensos para serem resolvidos em tempo real [44]. O modelo proposto por Desbrun *et al.* é baseado neste tipo de integração, porém utiliza uma aproximação, sendo necessária apenas a pré-computação de uma matriz constante, não comprometendo a execução em tempo real.

Sua única desvantagem aparente é com relação a objetos muito grandes (mais de 500 vértices), o que tanto torna a pré-computação da matriz demasiadamente lenta, como pode prejudicar a execução.

Porém, mesmo com estas potenciais quedas de desempenho ao se tentar aumentar a complexidade da simulação, há pesquisas na área sobre novas tecnologias que possam auxiliar no aumento do desempenho de tais aplicações, mesmo com um elevado nível de complexidade. Tais tecnologias englobam a utilização de *hardware* dedicado, como os exemplos citados no Capítulo 3.

Enquanto isso, novas tecnologias surgem, expandindo as possibilidades quanto à melhoria na execução de técnicas já conceituadas e avaliadas, como as apresentadas ao longo deste trabalho. Como exemplo de novas tecnologias, há a utilização da computação paralela massiva. Um exemplo de utilização deste paradigma é a *General Purpose Computation on Graphics Processing Unit* (GPGPU) [47], que utiliza todo o poder de processamento paralelo de um processador gráfico

para propósito geral a fim de acelerar a execução de aplicações com altíssimo custo computacional.

A tendência é que, com a expansão da utilização de tecnologias como estas, técnicas como as apresentadas neste trabalho poderão ser ainda mais incrementadas, tornando-se cada vez mais precisas e rápidas.

6.1. Contribuições

A implementação das técnicas descritas neste trabalho, propiciou uma análise comparativa, que demonstra que dentre os dois algoritmos, o PBD é mais eficiente para aplicações em tempo real, principalmente quando forem necessários muitos elementos ou elementos com grande quantidade de vértices. Aliado a isso, a sua complexidade computacional se eleva em menos intensidade que o modelo de Desbrun *et al.* quando a complexidade dos objetos aumenta.

Este trabalho também demonstra essencialmente que a capacidade de processamento das máquinas mais modernas torna possível a execução eficiente de aplicações gráficas, mesmo utilizando motores gráficos de alto nível, como o OGRE.

Este Trabalho de Graduação também será inserido no contexto de um projeto maior, que necessita da utilização de objetos maleáveis chamado VisGas, do Grupo de Pesquisa em Realidade Virtual e Multimídia – GRVM.

Finalmente, um artigo enfocando o desenvolvimento do PBD foi publicado no Workshop de Realidade Virtual e Aumentada de 2007 (WRVA 2007) [48] e uma submissão ao X Symposium on Virtual and Augmented Reality de 2008 (SVR 2008) foi realizada.

6.2. Trabalhos futuros

Como trabalhos futuros, podem-se citar a implementação de mais restrições, além da correção de alguns dos problemas encontrados no PBD. Este trabalho deverá ser integrado a um ambiente maior de simulação com demanda de utilização de objetos deformáveis, como descrito na seção anterior.

A criação de elementos tetraédricos, para avaliar a variação no desempenho e no comportamento físico dos objetos volumétricos, também está previsto. Estes novos elementos serão criados a partir de algoritmos já existentes de geração de modelos

tetraédricos, possivelmente baseados no modelo de triangulação de *Delaunay* como [49] ou [50], por exemplo.

À medida que os objetos e os cálculos forem ficando mais complexos, será necessário agir na direção de melhorar o desempenho na execução deste tipo de aplicação. Dessa forma, estudos na área de GPGPU serão realizados para integrar uma das técnicas detalhadas neste trabalho a este novo paradigma de computação.

Referências Bibliográficas

[1] BURDEA, G. C.; COIFFET, P. *Virtual Reality Technology*. 2ª edição, New Jersey, John Wiley & Sons, 2003.

[2] MÜLLER, M., HEIDELBERGER, B., TESCHNER, M., GROSS, M., Meshless Deformations Based on Shape Matching. *ACM Computer Graphics 2005*. ACM Press, Los Angeles, pp. 471-478, 2005.

[3] LASSETER, J. Principles of Traditional Animation Applied to 3D Computer Graphics. *ACM Computer Graphics 1987*, ACM Press, v. 21, n. 4, pp. 35-44, jul. 1987.

[4] TERZOPOULOS, D., PLATT, J., BARR, A., FLEISCHER, K. Elastically Deformable Models. *ACM Computer Graphics 1987*, ACM Press, v. 21, n. 4, pp. 205-214, jul. 1987.

[5] MOLINO, N. P. *Mesh Generation and Fracture for Deformable Bodies*. Stanford: Stanford University, 2004, 137p. Ph.D. Thesis.

[6] KÜHNAPFEL, U., ÇAKMAK, H. K., MAAß, H. 3D Modeling for Endoscopic Surgery. *IEEE Symposium on Simulation*, Delft, pp. 22-32, 1996.

[7] Crysis. Disponível: *EA: Crysis*. URL: http://www.ea.com/crysis/, consultado em: Janeiro de 2008.

[8] NEALEN, A., MÜLLER, M., KEISER, R., BOZERMAN, E., CARLSON, M. Physically Based Deformable Models in Computer Graphics. *State of The Art Report in proceedings of Eurographics*, pp. 71-94, 2005.

[9] MIRTICH, B., CANNY, J. Impulse Based Simulation of Rigid Bodies. *proceedings of Symposium on Interactive 3D Graphics*, pp. 181-188, 1995.

[10] MÜLLER, M., HEIDELBERGER, B., HENNIX, M., RATCLIFF, J. Position Based Dynamics. *in proceedings of VRIPhys*, Madrid, pp. 71-80, 2006.

[11] MILENKOVIC, V. J. Position-Based Physics: Simulating the Motion of Many Highly Interacting Spheres and Polyhedra. *ACM Computer Graphics 1996*, ACM Press, v. 30, pp. 129-136, 1996.

[12] BESBRUN, M., SCHRÖDER, P., BARR, A. Interactive Animation of Structured Deformable Objects. *Conference on Graphics Interface*, Morgan Kaufmann Publishers Inc., Kingston, pp. 1-8, 1999.

[13] STRANG, G. *Calculus*. 2ª edição, Wellesley, Wellesley-Cambridge Press, 1991.

[14] WEISSTEIN, E. W. Gradient. Disponível: *Mathworld – a Worlfram Web Resource*. URL: http://mathworld.wolfram.com/Gradient.html, consultado em Janeiro de 2008.

[15] WEISSTEIN, E. W. Runge-Kutta Method. Disponível: *Mathworld – a Wolfram Web Resource*. URL: http://mathworld.wolfram.com/Runge-KuttaMethod.html, consultado em Janeiro de 2008.

[16] ESI Group. Disponível: *ESI Group announces the latest version of PAM-CRASH 2G – ESI Group.* URL: http://www.esi-group.com /News/PAM-CRASH%202G%20V2007, consultado em: Janeiro de 2008.

[17] JAMES, D. L., PAI, D. K., ArtDefo: Accurate Real Time Deformable Objects. *ACM Computer Graphics 1999*, ACM Press/Addison-Wesley Publishing Co., Los Angeles, pp. 65-72, jul. 1999.

[18] FARIAS, T., SILVA, D., MOURA, G., BUENO, M., TEICHRIEB, V., KELNER, J. Ageia PhysX. *IX Symposium on Virtual and Augmented Reality*, Petrópolis, 2007.

[19] Open Dynamics Engine. Disponível: *Open Dynamics Engine – Home*. URL: http://www.ode.org/, consultado em: Janeiro de 2008.

[20] Ageia PhysX. Disponível: *About AGEIA PhysX*. URL: http://www.ageia.com/physx/ index.html, consultado em: Janeiro de 2008.

[21] 3D Studio Max. Disponível: *Autodesk 3ds max*. URL: http://usa.autodesk.com/adsk/servlet/ index?id=5659302&siteID=123112, consultado em: Janeiro de 2008.

[22] Maya. Disponível: *Autodesk Maya*. URL: http://usa.autodesk.com/adsk/servlet/ index?siteID=123112&id=7635018, consultado em Janeiro de 2008.

[23] Havok. Disponível: *Havok site*, URL: http://www.havok.com/, consultado em Janeiro de 2008.

[24] Havok Physics. Disponível: *Havok – Havok Physics*, URL: http://www.havok.com/ content/view/17/30/, consultado em: Janeiro de 2008.

[25] ELBERLY, D. H. *Game Physics (Interactive 3D Technology Series)*, Morgan Kaufmann Publishers, dec. 2003.

[26] GIBSON, S. F., MIRTICH, B. A Survey of Deformable Models in Computer Graphics. *Technical Report TR-97-19*, Mitsubishi Electric Research Laboratories, Cambridge, nov. 1997.

[27] SEDERBERG, T. W., PARRY, S. R. Free-form Deformation of Solid Geometric Models. *ACM Computer Graphics 1986*, ACM Press, v. 20, n. 4, pp. 151-160, aug. 1986.

[28] HSU, W. M., HUGHES, J. F., KAUFMAN, H. Direct Manipulation of Free-Form Deformations. *ACM Computer Graphics 1992*, ACM Press, v. 26, n. 2, pp. 177-184, 1992.

[29] KARNI, Z., GOTSMAN, C. Compression of soft-body animation sequences. *Computers & Graphics*, v. 28, n. 1, pp. 25-34, feb. 2004.

[30] BROWN, J., SORKIN, S., BRUYNS, C., LATOMBE, J.-C., MONTGOMERY, K., STEPHANIDES, M. Real-Time Simulation of Deformable Objects: Tools and Applications. *proceedings of 14th Conference on Computer Animation*, Seoul, pp. 228-258, 2001.

[31] BOSTROM, O., FILDES, B., MORRIS, A., SPARKE, L., SMITH, S., JUDD, R. A Cost Effective Far Side Crash Simulation. *International Journal of Crashworthiness*, v. 8, n. 3, pp. 307-313, jan. 2003.

[32] TERAN, J., BLEMKER, S., HING, V. N. T., FEDKIW, R. Finite Volume Methods for the Simulation of Skeletal Muscle. *ACM Computer Graphics/Eurographics Symposium on Computer Animation*, Eurographics Association, pp. 68-74, 2003.

[33] KHAREVYCH, L., KAHN, R. 3D Physics Engine for Elastic and Deformable Bodies. University of Maryland, College Park, dec. 2002.

[34] WEISSTEIN, E. W. Hooke's Law. Disponível: *Eric Weistein's World of Physics*, URL: http://scienceworld.wolfram.com/physics/HookesLaw.html, consultado em: Janeiro de 2008.

[35] JANSON, J., VERGEEST, J. S. M. A Discrete Mechanics Model for Deformable Bodies. *Computer-Aided Design*, v. 34, n. 2, pp. 913-928, oct. 2002.

[36] PLATT, J. C., BARR, A. H. Constraint Methods for Flexible Models. *ACM Computer Graphics 1988*, ACM Press, v. 22, n. 4, pp. 279-288, aug. 1988.

[37] GIBSON, S. F. F. 3D ChainMail: a Fast Algorithm for Deforming Volumetric Objects. *Symposium on Interactive 3D Graphics*, pp. 149-154, 195, nov. 1997.

[38] SUNDARAJ, K., MENDONZA, C., LAUGIER, C. A Fast Method to Simulate Virtual Deformable Objects with Force Feedback. *7th International Conference on Control, Automation, Robotics and Vision*, IEEE Press, v. 1, pp. 413-418, dec. 2002.

[39] COSTA, I. F., BALANIUK, R. Static Solution for Real Time Deformable Objects with Fluid Inside, *European Research Consortium for Informatics and Mathematics (ERCIM) News*, v. 44, pp. 44-45, jan. 2001.

[40] DEBUNNE, G., DESBRUN, M., CANI, M.-P., BARR, A. Adaptative Simulation of Soft Bodies in Real-Time. *Computer Animation 2000*, pp. 15-22, 2000.

[41] RANZUGLIA, G., GIGNONI, P., GANOVELLI, F., SCOPIGNO, R. Implementing Mesh-Based Approaches for Deformable Objects on GPU. *Fourth Eurographics Italian Chapter*, Eurographics Association, pp. 213-218, feb. 2006.

[42] MOSEGAARD, J., SØRENSEN, T. S. Real-Time Deformation of Detailed Geometry Based on Mappings to a Less Detailed Physical Simulation on GPU. *11th Eurographics Workshop on Virtual Environments*, Eurographics Association, pp. 105-111, 2005.

[43] JAKOBSEN, T. Advanced Character Physics, Disponível: *gamasutra.com*, URL: www.gamasutra.com/resource_guide/20030121/jacobson_01.shtml, consultado em: Janeiro de 2008.

[44] WITKIN, A., BARAFF, D., KASS, M. An Introduction to Physically Based Modeling: An Introduction to Continuum Dynamics for Computer Graphics. ACM Computer Graphics Course Notes Online Version, Disponível: *An Introduction to Physically Based Modeling*, URL: http://www.cs.cmu.edu/~baraff/pbm/pbm.html, consultado em: Janeiro de 2008.

[45] PROVOT, X. Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior. *Graphics Interface*, Canadian Human-Computer Communications Society, pp. 147-154, 1995.

[46] OGRE, Disponível: *OGRE Team site*, URL: http://www.ogre3d.org, consultado em Janeiro de 2008.

[47] GPGPU, Disponível em: *GPGPU site*, URL: http://www.gpgpu.org, consultado em Janeiro de 2008.

[48] ALMEIDA, M. W. S., ALMEIDA, G. F. de, TEICHRIEB, V., KELNER, J. Simulação Física de Objetos Deformáveis Através de um Algoritmo Baseado em Posição. *Workshop de Realidade Virtual e Aumentada 2007*, Itumbiara - GO, nov. 2007.

[49] MAUR, P., KOLINGEROVÁ, I. Post-optimization of Delaunay Tetrahedrization. *17th Spring Conference on Computer Graphics*, IEEE Computer Society, pp. 31, 2001.

[50] SAPIDIS, N. S., PERUCCHIO, R. Domain Delaunay Tetrahedrization of Arbitrarily Shaped Curved Polyhedra Defined in a Solid Modeling System. *ACM Symposium on Solid Modeling Foundations and CAD/CAM Applications*, ACM Press, pp. 465-480, 1991.

Judith Kelner

Veronica Teichrieb

Mozart William Santos Almeida