Ambiente Virtual Interativo com Colisão e Deformação de Objetos para Treinamento Médico

Marcello Kera [1]
Hélio Pedrini [2]
Fátima L. S. Nunes [3]

Resumo: O treinamento de procedimentos médicos pode ser beneficiado com o uso de ambientes virtuais interativos que simulam com realismo as ações do usuário. A simulação deve emitir respostas rápidas relativas ao encontro de objetos, deformação, restrição de movimento ou mesmo produzir forças e vibrações. Este trabalho descreve uma metodologia para criação de um ambiente virtual para treinamento médico. Classes e métodos são projetados e implementados no ambiente por meio da linguagem de programação Java. Métodos de colisão e deformação de objetos são utilizados para incorporar realismo à cena, sendo itens complexos e dependentes das informações de interação monitoradas no ambiente virtual. Para isso, um método híbrido de detecção de colisão foi desenvolvido para permitir concomitantemente precisão em relação à definição da existência de interpenetração entre os objetos e tempo de resposta adequado para que o usuário não perceba atrasos no sistema. Os objetos modelados são representados por malhas poligonais. A detecção de colisão entre objetos é baseada na subdivisão hierárquica do espaço com *octrees* e detecção de faces. A técnica de deformação massa-mola é utilizada para simular a alteração na forma dos objetos que se colidem. Experimentos são realizados para demonstrar as funcionalidades do protótipo.

[1]Departamento de Informática, Universidade Federal do Paraná (UFPR)
Curitiba-PR, 81531-990
[2]Instituto de Computação, Universidade Estadual de Campinas (UNICAMP)
Campinas-SP, 13084-971
[3]Escola de Artes, Ciências e Humanidades, Universidade de São Paulo (USP)
São Paulo-SP, 03828-000

Abstract: Medical procedure training may benefit from the use of interactive virtual environments that simulate realistically user actions. The simulation must provide fast responses related to object collision, deformation, movement constraints or even produce forces and vibrations. This paper describes a methodology for creating a virtual environment for medical training. Classes and methods are designed and implemented in the environment through the Java programming language. Object collision and deformation methods are used to incorporate realism to the scene, which are complex and dependent issues on interaction information tracked in the virtual environment. In order to do that, a hybrid collision detection method has been developed to allow both precision regarding the existence of interpenetration between the objects and adequate response time, so that the user does not notice delays in the system. The modeled objects are represented by polygonal meshes. The collision detection between objects is based on hierarchical subdivision of space with octrees and face detection. The mass-spring deformation technique is used to simulate changes in the shape of the objects that collide. Experiments are conducted to demonstrate the functionality of the prototype.

1 Introdução

Realidade Virtual (RV) pode ser definida como uma interface natural e poderosa entre homem e máquina, por permitir ao usuário interação, navegação e imersão em um ambiente tridimensional (3D) sintético, gerado por computador, por meio de canais multissensoriais, tais como visão, audição e tato [19]. De acordo com Hancock [10], a RV é a forma mais avançada de interface do usuário com o computador, envolvendo três premissas básicas: interação, envolvimento e imersão [26].

A imersão deve proporcionar ao usuário a sensação de presença dentro do mundo virtual. A RV pode ser considerada imersiva ou não imersiva. Na RV imersiva, capacetes ou cavernas (salas em que paredes, teto e chão são telas de projeção) são utilizados, enquanto a não imersiva utiliza apenas monitores de vídeo. Há também diferentes graus de imersão, levando em consideração dispositivos baseados em outros sentidos, como a audição e o tato. A interação está ligada à influência das ações do usuário no comportamento dos objetos, ou seja, o Ambiente Virtual (AV) é modificado de acordo com os comandos do usuário. O envolvimento, por sua vez, está associado ao grau de motivação que o mundo virtual proporciona a este usuário, podendo ser passivo a esse AV como, por exemplo, apenas a leitura de algo, ou ativo, como participar de um jogo.

A RV fornece mecanismos mais adequados para construção de aplicações que exigem interação avançada com o usuário, sendo uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos [2]. A RV é considerada

um paradigma no qual se usa um computador para a interação com algo irreal, mas que pode ser considerado real no momento em que está sendo utilizado [35].

A grande vantagem desse tipo de interface é que o conhecimento intuitivo do usuário a respeito do mundo físico pode ser utilizado para manipular o mundo virtual. Dispositivos não convencionais como capacetes, luvas de dados e dispositivos hápticos podem ser utilizados para a visualização desse mundo, permitindo que o usuário tenha uma sensação de estar em um mundo real [35].

Na RV, a renderização deve ser feita em tempo real, isto é, imagens do AV têm que ser atualizadas sempre que ocorrer uma modificação na cena [35]. Na prática, a RV faz com que o usuário navegue e observe um mundo 3D, em tempo real e com seis graus de liberdade, referentes a seis tipos de movimento: para frente/para trás, acima/abaixo, esquerda/direita, inclinação para cima/para baixo, angulação à esquerda/à direita e rotação à esquerda/à direita. Essencialmente, a RV é uma cópia da realidade física, na qual o indivíduo existe em um mundo 3D e tem a capacidade de interagir com o mundo ao seu redor em tempo real. Os equipamentos de RV (dispositivos não convencionais) simulam essas condições, em que o usuário pode até mesmo "tocar" os objetos de um mundo virtual e fazer com que eles respondam ou sofram alterações de acordo com suas ações [37].

Morie [25] define um AV como um ambiente 3D ou espaço imaginário gerado por computador. Em geral, os ambientes virtuais visam reproduzir situações do mundo real ou próximas daquelas percebidas no mundo real. Eles têm a finalidade de permitir simulação, treinamento, visualização ou outro tipo de atividade por meio de técnicas de RV.

Normalmente, os AVs construídos para aplicações de simulação e treinamento são compostos por dois ou mais objetos que podem se movimentar a partir da interação do usuário. De modo a permitir maior realismo às aplicações, os AVs normalmente devem prever características inerentes ao mundo real como movimentos, colisões e deformações. Vários domínios de conhecimento podem ser beneficiados pela Realidade Virtual, tais como medicina, educação, entretenimento, treinamento, visualização de informação, auditórios virtuais e artes.

O principal objetivo deste trabalho consiste na criação de um protótipo de AV para treinamento médico. Duas classes principais são projetadas e implementadas no ambiente com a linguagem de programação Java. Métodos de detecção de colisão entre objetos são estudados e avaliados, buscando-se uma solução com alto nível de precisão e bom desempenho, de forma a permitir uma simulação com realismo. Além disso, uma vez que os objetos que colidem podem sofrer alterações em suas formas, métodos de deformação são investigados e implementados, em particular, aqueles que utilizam malhas poligonais para representação dos objetos. Experimentos são realizados para demonstrar as funcionalidades do protótipo. As classes e os métodos desenvolvidos serão posteriormente integrados no *framework* ViMeT

(*Virtual Medical Training*) [27, 28].

O ambiente desenvolvido neste trabalho visa à simulação do procedimento de punção, o qual consiste na extração de pequenas partes de tecidos do órgão em questão para auxiliar a elaboração do diagnóstico médico. O trabalho inclui o estudo, a proposição, a implementação e a avaliação de métodos para colisão e deformação de objetos representados com malhas poligonais. O desenvolvimento do protótipo adota ferramentas computacionais de baixo custo, com tecnologia aberta, orientada a objetos, livre e multiplataforma, o que traz benefícios para a comunidade de usuários por facilitar a utilização e expansão de suas funcionalidades e por permitir a integração das aplicações existentes no *framework*.

Apesar de existirem alguns ambientes virtuais destinados ao treinamento médico, o ambiente aqui apresentado visa especificamente à simulação de exames de biópsia, considerando para isso técnicas de detecção de colisão e deformação com precisão e tempo de resposta adequados, características que não estão presentes em ambientes mais genéricos de simulação. Além desses aspectos, outra vantagem é geração de ferramentas com baixo custo, com independência de plataforma e que permitem o reuso de código, visto que as tecnologias utilizadas são gratuitas e utilizam o paradigma de orientação a objetos.

É importante salientar, ainda, que ambientes virtuais para treinamento médico, como é o caso do protótipo aqui descrito, permitem a repetição do treinamento ilimitadamente sem custos de manutenção, como ocorrem com os laboratórios físicos [22]. Além disso, verifica-se que o uso de simuladores virtuais permitem efetivamente a transferência das habilidades treinadas para situações reais, conforme assegurado em estudos anteriormente conduzidos [14, 16].

O restante do trabalho é dividido como segue. A seção 2 apresenta uma revisão dos principais fundamentos e abordagens relacionados à análise de detecção de colisão e deformação de objetos, além de um estudo de ferramentas para desenvolvimento do AV interativo. A metodologia empregada neste trabalho é descrita na seção 3, onde são ilustradas as etapas para a construção do protótipo. A seção 4 apresenta os resultados obtidos com a aplicação da metodologia. Finalmente, a seção 5 apresenta a conclusão do trabalho e considerações finais, bem como propostas para trabalhos futuros.

2 Conceitos e Trabalhos Relacionados

A fim de fornecer conceitos básicos para a compreensão da metodologia e dos resultados do presente artigo, esta seção apresenta conceitos sobre detecção de colisão, deformação e trabalhos relacionados com os assuntos aqui abordados.

2.1 Detecção de Colisão

Detectar uma colisão é verificar a aproximação entre objetos de um AV. A aproximação desses objetos deve ser suficientemente pequena a ponto de possibilitar a ocorrência de uma sobreposição entre eles. A sua percepção exige a presença de, no mínimo, dois objetos em um AV, sendo que pelo menos um deles deve estar em movimento [36].

Trata-se de um problema complexo, visto que objetos virtuais podem ter formas, tamanhos e movimentos variados, dependendo de suas naturezas e da finalidade da aplicação. Além disso, os objetos em cena podem ser rígidos ou deformáveis.

A detecção de colisão é um dos itens mais complexos e dependentes das informações de interação monitoradas em um AV, permitindo responder às interações entre objetos no mundo virtual, fator importante para obtenção do realismo. Esse requisito exige programas específicos para gerenciar as informações de interação, envolvendo rotinas de controle e computação gráfica [23].

Devido ao fato de que a detecção de colisão depende extremamente das informações provenientes das interações do usuário com o AV, algoritmos eficientes devem ser desenvolvidos para alcançar precisão dentro de tempos de respostas aceitáveis. Um conjunto de abordagens de algoritmos de detecção de colisão existentes na literatura é apresentado sucintamente a seguir.

Volumes Limitantes

A detecção de colisão por volumes limitantes consiste em utilizar primitivas simples para delimitação do espaço de colisão do objeto. A principal ideia desse tipo de abordagem é verificar os pontos mais afastados do objeto e inserir a primitiva nesse espaço. Nesta categoria é interessante citar o algoritmo de caixas envoltórias alinhadas aos eixos (*Axis Align Bounding Boxes - AABB*), que verifica os dois pontos mais afastados do objeto e envolve esse objeto com uma caixa, alinhando-a com os eixos de coordenadas do sistema [20, 36].

Outro algoritmo interessante é o que emprega caixas orientadas ou OBB (*Oriented Bounding Box*). A OBB é definida pelos seguintes atributos: centro, três vetores unitários que representam a direção da caixa e a extensão em cada direção. A OBB verifica os pontos mais afastados do objeto para definir seu volume limite e faz com que esse volume se oriente pela forma do objeto e se encaixe de maneira que sua área fique a mais próxima possível da forma do objeto [8].

Similar ao algoritmo AABB, o método de esferas (*spheres*) testa os pontos mais afastados do objeto e envolve o mesmo com uma esfera. Esferas envolventes são representadas por um centro e um raio. Uma esfera é definida pelo conjunto de todos os pontos x equidistantes de um ponto central com uma distância $r > 0$. Para um objeto geométrico que consiste de uma coleção de pontos $(V_i)_{i=0}^{n}$, uma esfera envolvente pode ser calculada de várias formas. A

vantagem desse método é que ele é independente da orientação, em contraste com o método de caixas que requer o alinhamento dos eixos. A desvantagem é que esse método pode não envolver adequadamente um objeto longo e fino, o que pode resultar em falsa detecção de colisão.

Subdivisão Hierárquica do Espaço

Nesta categoria, o ambiente é subdividido em um espaço hierárquico. Objetos no ambiente são agrupados de acordo com a região em que se encontram. Quando um objeto no ambiente se movimenta e troca de posição, movendo-se para uma outra região do espaço, apenas os objetos do novo espaço precisam ser verificados se colidem ou não com o objeto em movimento.

O ambiente virtual pode ser subdividido utilizando-se alguns métodos conhecidos tais como *octrees*, *K-d trees* e *BSP trees*. Segundo Hearn e Baker [11], *octrees* são estruturas de árvores hierárquicas em que cada nó interno tem até oito filhos e é organizada para que cada nó corresponda a uma região do espaço tridimensional. O esquema de codificação divide regiões de espaços tridimensionais, normalmente cubos, em octantes e armazena elementos em cada nó da árvore.

Outros tipos de árvores são citadas na literatura como árvores k-dimensionais [4, 6] e BSP [34]. Como não estão diretamente relacionadas ao escopo deste trabalho, os detalhes de sua implementação podem ser encontrados em [17, 18].

Subdivisão Hierárquica do Objeto

Nesta categoria, o objeto é subdividido em regiões, sendo que a detecção de colisão é feita quando uma dessas regiões é interceptada por um outro objeto. A hierarquia de volumes envolventes é baseada em dois tipos de nós: nós internos e nós folhas. Cada nó, independentemente do tipo, possui um volume envolvente associado. Nós folhas possuem volumes envolventes que englobam somente a geometria (ou seja, a malha de triângulos) enquanto nós internos possuem volumes envolventes que englobam todos os volumes envolventes de seus filhos. Por isso, pode-se descartar todos os filhos se o volume envolvente do nó pai não colidir com outro nó. Outro aspecto muito importante da hierarquia é a noção de espaço. Como todos os nós e seus volumes envolventes são definidos no espaco local do objeto, surge a necessidade de associar a cada nó uma matriz de transformação responsável por mapear este nó para o espaço do nó pai. A detecção de colisão é efetuada no espaço global.

Uma forma de se implementar a subdivisão hierárquica do objeto é utilizar o método de árvores de volume limite (AVL) [20], onde cada nó n corresponde a um subconjunto $C.n$ pertencente a C, com o nó raiz sendo associado a um volume limite com o total do conjunto C. Cada nó interno tem dois ou mais filhos, sendo que o número máximo de filhos para cada nó interno é chamado de grau da árvore. A união de todos os subconjuntos associados com o

filho do conjunto n é igual a $C.n$. Cada nó é associado também a um volume limite $v(C)$, que é uma aproximação do espaço ocupado por $C.n$, usando representações mínimas de formas, como caixas e esferas.

Os métodos mais conhecidos e que utilizam AVL são: *sphere-trees* [13], *AABB-trees* [36], *OBB-trees* [8] e *k-dops* [20]. Há outros algoritmos conhecidos e eficientes encontrados na literatura, mas são variações desses métodos fundamentais utilizados em detecção de colisão.

Computação Incremental da Distância: a distância mínima é verificada incrementalmente entre um par de objetos, ou seja, a cada iteração do ambiente é feita uma verificação da distância entre os objetos. Quando essa distância se torna tão pequena a ponto de causar a colisão entre objetos, há uma notificação ao ambiente sobre a detecção de colisão.

Há vários métodos propostos para a computação incremental da distância, bem como para a computação hierárquica da distância para corpos convexos em movimento [9], algoritmos eficientes para computação da distância incremental [21], algoritmos incrementais para detecção de colisão entre polígonos [30], entre outros.

2.2 Deformação

A deformação de objetos é um recurso que auxilia a transformação de um AV em uma simulação mais próxima do mundo real. Em objetos 3D, a deformação é implementada em modelos que permitam a manipulação de sua estrutura (vértices e arestas) [5]. Ela consiste basicamente na mudança da estrutura poligonal de um modelo cujo resultado final é um novo modelo após a aplicação de uma tensão ou uma variação térmica no objeto.

Métodos de deformação têm sido propostos e as três técnicas mais citadas na literatura são: Deformação de Forma Livre [7, 12], Métodos de Elementos Finitos [24] e Massa-Mola [31], sucintamente descritas a seguir.

Deformação de Forma Livre

A Deformação de Forma Livre (*Free Form Deformation* - FFD) consiste em deformar modelos geométricos sólidos de uma maneira livre através de pontos de controle. Simplificadamente, a deformação de forma livre consiste em envolver um objeto de qualquer representação gráfica por um volume parametrizado. Uma ligação entre o objeto e os pontos de controle do volume parametrizado é então realizada [24]. Choi et al. [5] lembram que os movimentos de controle dos pontos e as correspondentes mudanças na estrutura dos objetos devem estar ligados para obter o resultado de deformações esperado. Mesmo sendo versátil, esse método apresenta dificuldades para limitar as deformações para pequenas regiões, pois somente os objetos inseridos dentro das grades são deformados globalmente. Como o método

FFD desconsidera propriedades físicas para deformação de objetos, essa técnica acaba sendo limitada e custosa, pois não permite a manipulação dos objetos que compõem a cena, não sendo uma técnica satisfatória para aplicações de RV.

Método de Elementos Finitos

O método de Elementos Finitos (*Finite Element Method* - FEM) utiliza técnicas matemáticas para encontrar soluções aproximadas para equações diferenciais parciais. A técnica subdivide o domínio do problema, resultando em partes menores de malhas poligonais. Tem por objetivo reduzir complicadas equações diferenciais em um grupo de equações algébricas que podem ser solucionadas numericamente [24].

Massa-Mola

O método Massa-Mola (*Mass Spring - MS*) está relacionado aos sistemas do tipo massa-mola considerados na área de Mecânica. Permite a remodelagem de objetos através de pontos de massa conectados por molas. Cada um dos pontos de massa é o mapeamento de um ponto específico do objeto. O deslocamento desses pontos descreve a deformação do objeto. A mola é uma estrutura que possui elasticidade permitindo a sua deformação quando uma pressão é exercida sobre ela; possui uma força linear e é baseada na lei de Hooke [31]. A força produzida pela mola é diretamente proporcional ao seu deslocamento do estado inicial. A equação 1 denota o cálculo da deformação:

$$F = k\,\Delta l \tag{1}$$

em que F é a força elástica (Newton), k é uma constante positiva denominada constante elástica da mola (Newton/metro) e Δl é a deformação da mola (metro). A constante elástica traduz a rigidez da mola. Quanto maior for a constante elástica da mola, maior será sua resistência.

Essa expressão é sempre calculada quando a mola sai de seu estado de equilíbrio (estado natural da mola) e passa a ser comprimida ou esticada. A lei de Hooke pode ser utilizada desde que o limite elástico do material não seja excedido. O comportamento elástico dos materiais segue o regime elástico na lei de Hooke apenas até um determinado valor de força. Após este valor, a relação de proporcionalidade deixa de ser definida. Se essa força continuar a aumentar, o corpo perde a sua elasticidade e a deformação passa a ser permanente (inelástico), chegando a romper o material.

2.3 Ferramentas JOGL e Java3D

Com o crescimento da área de RV, ferramentas de auxílio ao desenvolvimento de aplicações têm surgido nos últimos anos e várias delas têm sido disponibilizadas gratuitamente para que aplicações de baixo custo sejam produzidas. Entre elas, pode-se citar GL4Java,

Java3D, LWJGL (*Lightweight Java Game Library*) e JOGL. Para a análise descrita, apenas Java3D e JOGL foram consideradas por haver um conjunto muito grande de ferramentas possíveis de serem utilizadas e também porque o *framework* de teste foi escrito em Java3D.

Java3D

A API Java3D (J3D) consiste em uma hierarquia de classes Java que serve como interface para o desenvolvimento de sistemas gráficos tridimensionais de renderização [18]. Ela possui construtores de alto nível que permitem a criação e manipulação de objetos geométricos, especificados em um universo virtual. J3D também possibilita a criação de universos virtuais com uma grande flexibilidade, em que as cenas são representadas por meio de grafos e os detalhes de sua visualização são gerenciados automaticamente [32, 33].

Um grafo de cena é uma estrutura de dados que descreve uma cena tridimensional na forma de nós e arestas. Os nós guardam todas as propriedades dos objetos presentes numa cena, como geometria, cor, transparência e textura. As arestas guardam as relações entre os nós. Alguns motivos podem levar ao uso de J3D como: alto nível de abstração, importação direta de modelos criados com outras ferramentas para a sua representação interna, definição de som 3D, facilidade para utilização de dispositivos de RV, integração com a Internet e possibilidade de conexão com sistemas de banco de dados, além da gratuidade e portabilidade da linguagem Java [32, 33]. Assim, o desenvolvimento de um programa J3D se resume na criação de objetos e no seu posicionamento em um grafo de cena, que os combina em uma estrutura de árvore, podendo ser tratados tanto individualmente quanto em grupo. visualizado.

Na J3D, há três meios de se implementar um AV, os quais são chamados de *universos*: o *SimpleUniverse*, o *VirtualUniverse* e o *ConfiguredUniverse*. A classe *SimpleUniverse* permite que se crie facilmente um ambiente mínimo de usuário para um programa J3D ser executado. Para muitas aplicações básicas de J3D, a *SimpleUniverse* provê todas as funcionalidades necessárias às aplicações. Aplicações mais sofisticadas podem requerer mais controles para adquirir funcionalidades extras [32, 33].

JOGL

A API JOGL (Java OpenGL) provê a ligação entre a biblioteca gráfica OpenGL e a linguagem Java. Ela permite o acesso a muitas funcionalidades existentes na linguagem C, como a biblioteca de sistema de janelas GLUT, além de integrar facilmente as APIs Java que são padrões de gerenciamento de janelas como a AWT e a Swing [15].

Ela estabelece o GLCanvas como principal *widget* AWT, um componente que suporta aceleração por hardware e que tem por objetivo ser o *widget* primário utilizado em uma aplicação. O GLJPanel é o *widget* Swing que suporta aceleração por hardware.

Os conceitos apresentados nesta seção foram utilizados como embasamento teórico para o desenvolvimento das técnicas específicas para o AV apresentado na próxima seção. No

entanto, uma análise crítica se faz necessária a fim de enfatizar as necessidades específicas para treinamento médico.

Em relação à detecção de colisão, verifica-se que os métodos clássicos ou não apresentam a precisão necessária para as aplicações de treinamento médico ou exigem uma taxa de processamento extremamente elevada, inviável quando se deseja interação em tempo real. Assim, faz-se necessária a pesquisa a fim de definir métodos híbridos para treinamento médico, maximizando a relação entre precisão e tempo de resposta, como se propõe na próxima seção.

Os métodos clássicos de deformação apresentam problemas similares. O FFD não fornece a precisão necessária para treinamento médico e o FEM exige taxas de processamento inviáveis para execução em tempo real. Dentro os citados, o método Massa-Mola é o mais viável, entretanto, algumas adaptações são ainda necessárias a fim de permitir sua execução em tempo real, conforme será mostrado na próxima seção.

Finalmente, é interessante comentar a questão das tecnologias citadas. A linguagem Java, por concepção, permite que aplicações sejam executadas com independência de plataforma. Além disso, a sua gratuidade faz com que seja possível gerar ferramentas com baixo custo de produção e execução. O paradigma de orientação a objetos, seguido por esta linguagem, permite o reuso de código. Essa característica é muito interessante quando se trata de implementação de *frameworks* que visam fornecer um conjunto de classes capazes de gerar aplicações semiprontas dentro de um certo domínio (no caso deste trabalho, treinamento de exames de biópsia) permitindo a reutilização de projeto, arquitetura e codificação. Soma-se a isso, o fato de que as bibliotecas citadas – Java3D e JOGL – fornecem classes e métodos para disponibilização de ambientes virtuais, permitindo implementar características de Realidade Virtual sem grandes dificuldades.

3 Metodologia do Trabalho

Esta seção descreve a metodologia para criação do AV interativo para treinamento médico, incluindo as classes e os métodos para detecção de colisão e deformação de objetos, seus diagramas e pseudocódigos correspondentes.

3.1 Detecção de Colisão dos Objetos

Apesar do grande número de algoritmos para a detecção de colisão encontrados na literatura, não há ainda um método que combine satisfatoriamente dois aspectos importantes na detecção de colisão: precisão e desempenho.

Com a ideia de simular um ambiente de treinamento médico em um AV, como uma fer-

ramenta de baixo custo, há necessidade de se obter uma precisão que ofereça realismo, já que a ferramenta tem como objetivo treinar futuros profissionais da área da saúde. O desempenho da ferramenta deve também ser alto para permitir a simulação em tempo real dos eventos que ocorrem no AV.

O método de colisão empregado neste trabalho foi baseado no algoritmo desenvolvido por Antonio [1] e o método de deformação foi adaptado do *framework* ViMeT [27]. Para a construção do AV, utilizou-se o pacote JOGL [15].

O método de colisão compreende a divisão espacial, a computação da distância incremental e a interpolação de faces. A Figura 1 mostra um diagrama com as principais etapas do método de colisão desenvolvido, enumeradas a seguir:

- os objetos são inseridos na cena, ou seja, no AV;
- o AV é dividido em octantes, utilizando o conceito de divisão espacial (*octrees*);
- os objetos no AV são distribuídos nos octantes;
- a cada movimento do objeto é utilizada a computação incremental da distância para verificar se os objetos estão no mesmo octante;
- o teste de colisão é executado face a face.

Para carregar um objeto na cena, o ambiente desenvolvido utiliza modelos no formato OBJ [29] ou 3DS [3], que consistem em arquivos contendo informações básicas do modelo 3D: vértices, faces e normais. Com esses dados é possível reproduzir o modelo no ambiente. Os vértices são guardados em uma lista que contém as coordenadas espaciais x, y e z.

Para a lista de faces, apenas os índices das faces do objeto são armazenados. Por exemplo, se os vértices V_1, V_2, V_3 pertencem à face F_1 com V_1 na posição 1 da lista de vértices, V_2 na posição 2 e V_3 na posição 3, então a lista de faces recebe a seguinte informação: $F_1[1].x = 1$, $F_1[1].y = 2$ e $F_1[1].z = 3$. Assim, o armazenamento de informações é reduzido e também é eliminada a redundância de informações (coordenadas repetidas de vértices entre diferentes faces).

O cálculo das normais é efetuado para cada face dos objetos. O vetor normal (x_N, y_N, z_N) de cada face é armazenado em uma lista de normais. O cálculo das normais pode ser expresso como:

$$\begin{aligned} x_N &= (V_1.y\, V_2.z) - (V_1.z\, V_2.y) \\ y_N &= (V_1.z\, V_2.x) - (V_1.x\, V_2.z) \\ z_N &= (V_1.x\, V_2.y) - (V_1.y\, V_2.x) \end{aligned} \tag{2}$$

Após carregado o objeto na cena, esta é dividida em octantes. A divisão da cena é feita de acordo com a posição dos objetos, tal que os objetos mais afastados são detectados e

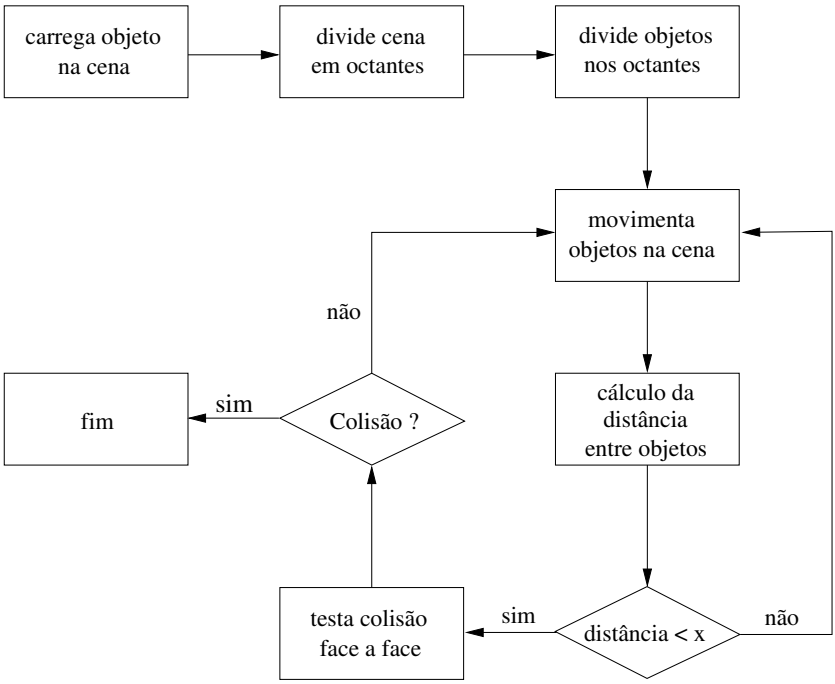

Figura 1: Diagrama ilustrando os passos do método de colisão.

suas coordenadas mínimas e máximas são usadas na subdivisão. A Figura 2 ilustra a divisão da cena. Após essa divisão, o posicionamento dos objetos é realizado em cada octante. Se um objeto ocupar mais do que um octante, ou seja, se as coordenadas que delimitam o objeto estiverem distribuídas em mais de um octante, suas faces são divididas entre os octantes.

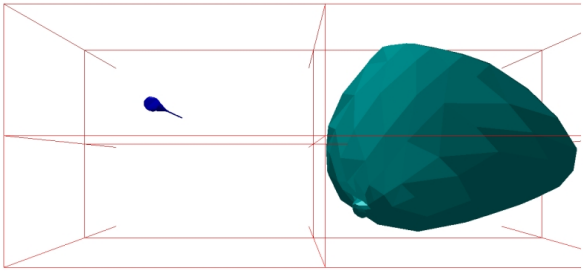

Figura 2: Divisão da cena utilizando *octrees*.

Ao movimentar um objeto na cena é possível verificar como esses octantes são dinamicamente alterados. Se dois objetos estiverem no mesmo octante, uma nova subdivisão do mesmo é feita até que um objeto ou suas faces não compartilhem esse octante com o outro objeto.

Para verificar a colisão entre os objetos, o método inclui os seguintes passos: verificação da coplanaridade dos triângulos, teste entre os pontos dos triângulos e teste de um ponto sobre o triângulo. Esse método foi baseado no algoritmo de Antonio [1]. Os pseudocódigos de 1 a 4 mostram os passos seguidos para a obtenção da colisão no AV.

```
início
    construirCena();
    enquanto RenderizaCena faça
        dividirCenaEmOctree();
        renderizarCena();
        verificarColisao();
    fim
fim
```

Algoritmo 1: Módulo principal.

```
início
    dividirCenaEmOctree() {
    calcularRaizOctree;
    subdividirOctree;
    posicionarFacesNaOctree;
    }
fim
```

Algoritmo 2: Módulo de divisão da cena e posicionamento de faces na *octree*.

```
início
    verificarColisao() {
    verificarAlturaMaximaOctree;
    verificarDiametroOctree;
    verificarColisaoFaceFace();
    }
fim
```

Algoritmo 3: Módulo de verificação de colisão.

```
início
    verificarColisaoFaceFace() {
    verificarCoplanaridade;
    testarArestasEntreTriangulos;
    testarPontoNoTriangulo;
    }
fim
```

Algoritmo 4: Módulo de colisão baseado em [1].

3.2 Deformação dos Objetos

A classe de deformação simula a alteração na forma de um objeto reposicionando os vértices do mesmo e de regiões vizinhas, usando a técnica massa-mola. Para a obtenção da fórmula de força da mola é preciso considerar que os vértices ou nós de massa do objeto estão conectados por mola [5].

O método de deformação é composto pelos passos ilustrados no diagrama da Figura 3:

- recebe faces em colisão;
- busca ponto mais próximo de colisão entre os objetos;
- procura pelos vértices imediatamente adjacentes ao ponto escolhido, ou seja, localizados na camada 1 (Figura 4);
- procura vizinhos da camada anterior; esse passo é realizado recursivamente até que o número de camadas definidas para a deformação seja atingida. A definição do número de camadas é feita pelo cálculo da força exercida sobre o ponto;
- deforma o objeto no ponto de colisão (muda a posição do vértice) e em suas camadas subsequentes.

A partir das faces obtidas no método de colisão, realiza-se o cálculo da distância Euclidiana entre os pontos das faces, tal que a menor distância é escolhida como vértice inicial para receber a deformação. Após essa escolha, uma pesquisa é feita na lista de faces buscando-se os vértices vizinhos do vértice raiz para a formação da primeira camada. A busca pelos vizinhos é feita recursivamente até ser atingido o número de camadas estabelecido.

Os vértices que estão diretamente conectados ao vértice raiz pertencem à primeira camada de deformação. A segunda camada é composta pelos vértices que estão conectados aos vértices da primeira camada, ou seja, vértices secundários ao vértice raiz e assim sucessivamente. As informações sobre quais vértices pertencem a quais camadas são armazenadas em uma estrutura de dados definida como *vizinhos*.

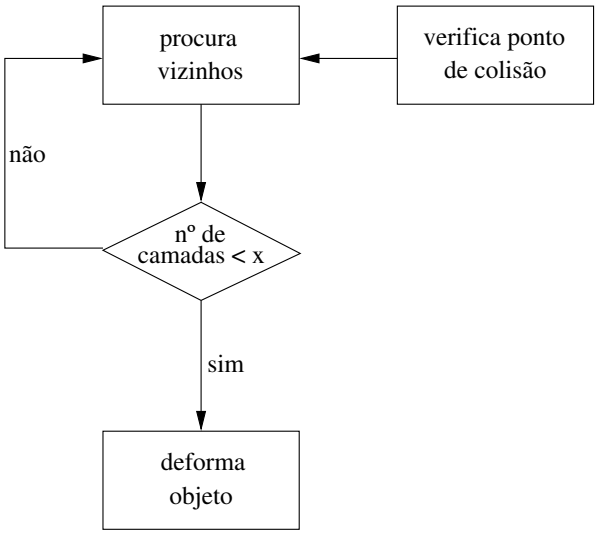

Figura 3: Diagrama ilustrando os passos do método de deformação.

A Figura 4 ilustra a escolha de vizinhos ao vértice raiz e a formação das camadas. O vértice raiz é representado pela cor azul, primeira camada em rosa, segunda camada em roxo e terceira camada em verde.

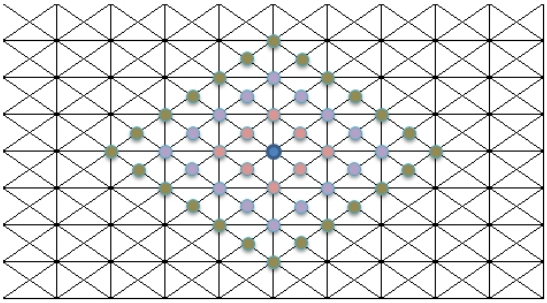

Figura 4: Formação de camadas na malha poligonal.

Ao ser aplicada uma força F no vértice raiz, o comportamento dinâmico deste é regido pelas equações de Newton (Segunda Lei de Newton, $F = ma$), em que F é a força aplicada, m é a massa e a é a aceleração. A aceleração é a derivada da velocidade v, em relação ao

tempo, que, por sua vez, é a derivada do espaço x em relação ao tempo.

$$a = \frac{dv}{dt} = \frac{d^2x}{dt^2} \tag{3}$$

Combinando a equação de Newton com a expressão da força, tem-se:

$$m\frac{d^2x}{dt^2} = -k\Delta t \tag{4}$$

Com isso, tem-se a determinação da variação sobre o tempo, implicando na posição final da mola em uma determinada variação do tempo. Para deformar um objeto, a equação 5 é utilizada:

$$m_i\frac{d^2u_i}{dt^2} + \alpha_i\frac{du_i}{dt} + \sum_{j\ \in\ \text{vértices conectados}} \frac{k_{ij}(\|r_{ij}\| - l_{ij})}{\|r_{ij}\|} r_{ij} = F_i \tag{5}$$

em que m_i, u_i e α_i são, respectivamente, a massa, posição e constante de amortecimento do vértice raiz. O vetor que representa a distância entre o vértice i e o vértice j é dado por $r_{ij} = u_j u_i$ e l_{ij} é o tamanho natural da mola que conecta o vértice i ao vértice j.

Usando o método de diferenças finitas, define-se Δt como um intervalo da variação do tempo. A posição do vértice i no instante $t + \Delta t$ é dada por $u_i(t + \Delta t)$ e esta é calculada tomando-se a posição do vértice i no instante corrente t e a sua posição no instante anterior $t - \Delta t$.

As constantes de massa-mola, força e amortecimento são inseridas manualmente. A quantidade de camadas afetadas na deformação depende diretamente da força exercida no vértice raiz e também dos parâmetros de constante da mola e amortecimento, uma vez que são eles que permitem uma modelagem matemática das estruturas físicas do objeto.

À medida em que os vértices são deslocados pela força externa, a força das molas afeta os vértices vizinhos. Assim, a deformação é propagada para as camadas, ou seja, quando um vértice se desloca da posição P para P', juntamente com ele são deslocados os demais vértices com os quais ele possui ligações.

Dessa forma, quando um ponto é deslocado, sua posição deve ser recalculada por meio do termo $\frac{k_{ij}(\|r_{ij}\|-l_{ij})}{r_{ij}} r_{ij}$ da equação 5, que é o cálculo da força de uma única mola, tendo como componentes específicos k que corresponde à constante da mola, r representando o vetor de distância entre os pontos i e j, e l que é o tamanho natural da mola. Este valor é utilizado para deslocar os vértices vizinhos e gerar uma alteração na posição do vértice, resultando no efeito de deformação desejado. Um valor maior que zero no resultado desse

cálculo indica que a força deve ser propagada para as próximas camadas; caso contrário, indica que a camada atual é a última a ser processada.

O algoritmo 5 apresenta as principais etapas do método de deformação.

```
início
    deformacao(face1, face2) {
    selecionaNoRaiz(face1, face2);
    selecionaVizinhos;
    calculaForca;
    enquanto Força > 0 faça
        selecionaCamada;
        selecionaVizinhos;
        calculaForca;
    fim
    }
fim
```

Algoritmo 5: Módulo de deformação.

4 Resultados Experimentais

Nesta seção são descritos os resultados da implementação dos métodos de colisão e deformação, incluindo as características da plataforma de desenvolvimento, os experimentos realizados, a captura das imagens e uma discussão acerca dos resultados obtidos.

A partir desses experimentos, tornou-se possível analisar o desempenho e a precisão dos métodos desenvolvidos e que foram incorporados à ferramenta.

4.1 Critérios para os Testes

Alguns parâmetros foram definidos para a realização dos testes. Em aplicações cuja interação é efetuada com dispositivos não convencionais, tais como luvas e capacetes, esses atributos podem ser dinamicamente ajustados.

No método de colisão foram realizados testes com os parâmetros diâmetro mínimo e altura máxima (valores que definem dimensões das caixas nas *octrees*), enquanto no método de deformação, os valores da força aplicada foram variados. Os valores para a constante da mola (K), constante de amortecimento (D) e a massa (M) permaneceram os mesmos ao longo de todos os testes.

Em aplicações que tenham interação com dispositivos não convencionais as características do objeto representado – tais como resistência, densidade e elasticidade – e da interação

com o usuário – tal como a força fornecida – são consideradas para definir esses atributos dinamicamente. Como nos testes realizados não foram utilizados dispositivos não convencionais, optou-se por usar valores empíricos para esses parâmetros. Adicionalmente, salienta-se que os valores utilizados para a detecção de colisão estabelecem o quanto a técnica deve ser precisa, tendo sido também definidos após diversos experimentos executados.

Os experimentos foram realizados de duas formas, um com apenas o método de colisão e o outro com a colisão e a deformação integradas. Como o ambiente possibilita a visualização dos objetos no ambiente com as malhas poligonais destacadas (*wireframe*) e esse recurso pode alterar a média de quadros por segundo (FPS, *frames per second*), os testes também mostraram a média de FPS com a visualização das malhas poligonais.

Duas plataformas diferentes foram testadas nos experimentos para avaliar o desempenho dos métodos: um computador com sistema operacional Windows Vista com processador Intel Core2Duo T5250 1.50 GHz, 3 GB de memória RAM e placa de vídeo Intel x3100 *on-board* e um computador com sistema operacional Linux Debian 2.6.18-6-686 com processador Intel Pentium 4 Xeon 3.20GHz, 2GB de memória RAM e placa de vídeo G-Force4 MX4000 de 64 bits.

4.2 Detecção de Colisão entre Objetos

Objetos para simulação do procedimento de punção em órgãos humanos foram empregadas nos testes de detecção de colisão. Inicialmente, dois objetos foram utilizados: um para representar um instrumento médico (uma seringa) e o outro para representar um órgão humano (uma mama). A quantidade de polígonos presentes na cena foi de 964 vértices e 1.817 faces para o objeto *seringa* e de 8.872 vértices e 17.490 faces para a *mama*. O número de vértices e faces é inerente aos modelos tridimensionais disponíveis para os testes. Quaisquer outros modelos poderiam ter sido empregados. É interessante salientar que quanto maior a quantidade de polígonos, mais comprometido poderia ficar o desempenho do sistema. Os resultados apresentados a seguir têm a intenção de mostrar que, mesmo com o grande número utilizado de vértices e faces, o sistema apresentou uma resposta bem satisfatória em termos de percepção de realismo.

A Figura 5(a) mostra os objetos em suas posições iniciais após serem carregados no ambiente. Inicialmente, a taxa de renderização está em torno de 28 FPS na plataforma Windows e de 27 FPS na plataforma Linux. A Figura 5(b) mostra os objetos na cena em posição inicial com a representação *octree*, na qual se pode notar que, mesmo estando em sua posição inicial, a aplicação já determina em qual octante cada objeto ou parte dele está inserido. Pode-se observar que, já neste primeiro momento, uma parte da mama estava no mesmo octante que a seringa. Com isso, esse octante foi dividido em outros 8 octantes, fazendo com que cada octante contenha apenas partes do mesmo objeto. A taxa média de renderização foi

compatível com a mostrada na cena anterior: 28 FPS na plataforma Windows e de 27 FPS na plataforma Linux. Ao apresentar os objetos com malhas poligonais visíveis, tem-se uma taxa média de 16 FPS na plataforma Windows e uma média de 15 FPS na plataforma Linux. A Figura 5(c) mostra a cena com os objetos em suas posições iniciais.

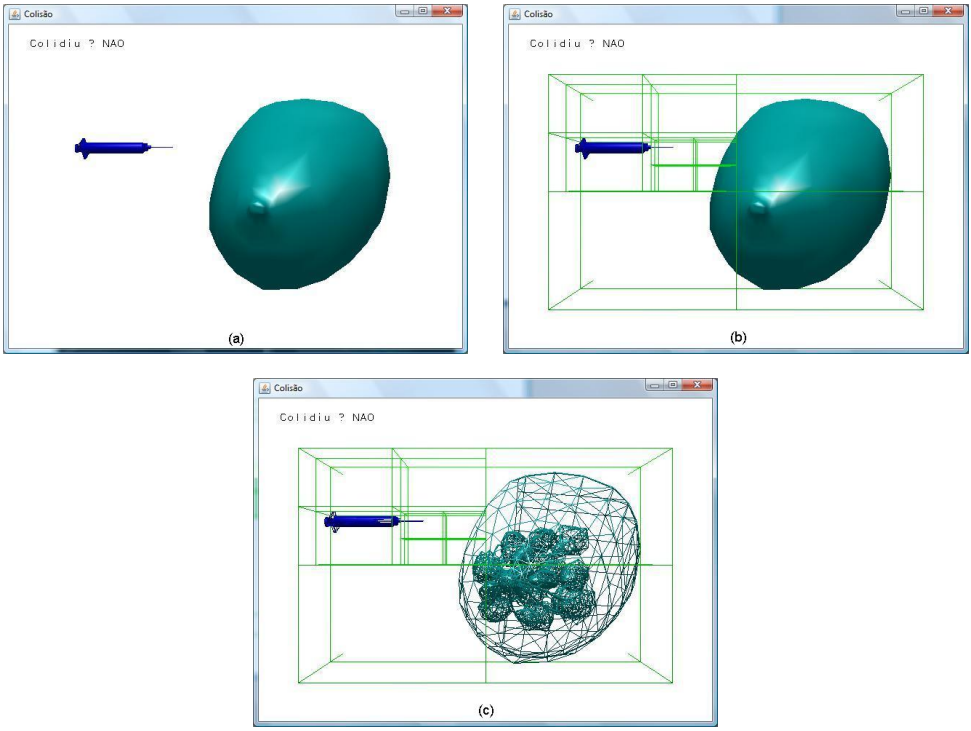

Figura 5: Objeto sob renderização. (a) em posição inicial; (b) em posição inicial sobreposto com *octree*; (c) em posição inicial com malhas poligonais visíveis.

Havendo uma colisão entre os objetos, verifica-se uma taxa de 24 FPS para a plataforma Windows e de 23 FPS para Linux. A Figura 6(a) ilustra o momento da colisão entre os objetos, verificada após a divisão recursiva dos octantes até que se atinja um tamanho mínimo do octante. Após essa divisão, o cálculo de sobreposição de faces é efetuado. Havendo colisão entre as faces ou se um valor pré-especificado de distância mínima for alcançado entre as faces, então a colisão é detectada. A Figura 6(b) mostra a cena com sobreposição da *octree* durante a detecção da colisão dos objetos, podendo ser observada a quantidade de octantes. A Figura 6(c) mostra uma região ampliada próxima ao ponto de colisão, detectado pelo método proposto. Nesta visualização, pode-se verificar a proximidade dos objetos, tendo então que a

colisão ocorre muito próxima à face do outro objeto. A Figura 6(d) mostra uma ampliação da região de detecção com as malhas poligonais visíveis. Nesta visualização pode ser observado com mais clareza que a extremidade do objeto *seringa* atingiu a distância mínima em relação ao objeto *mama*, sendo identificada uma colisão entre os objetos.

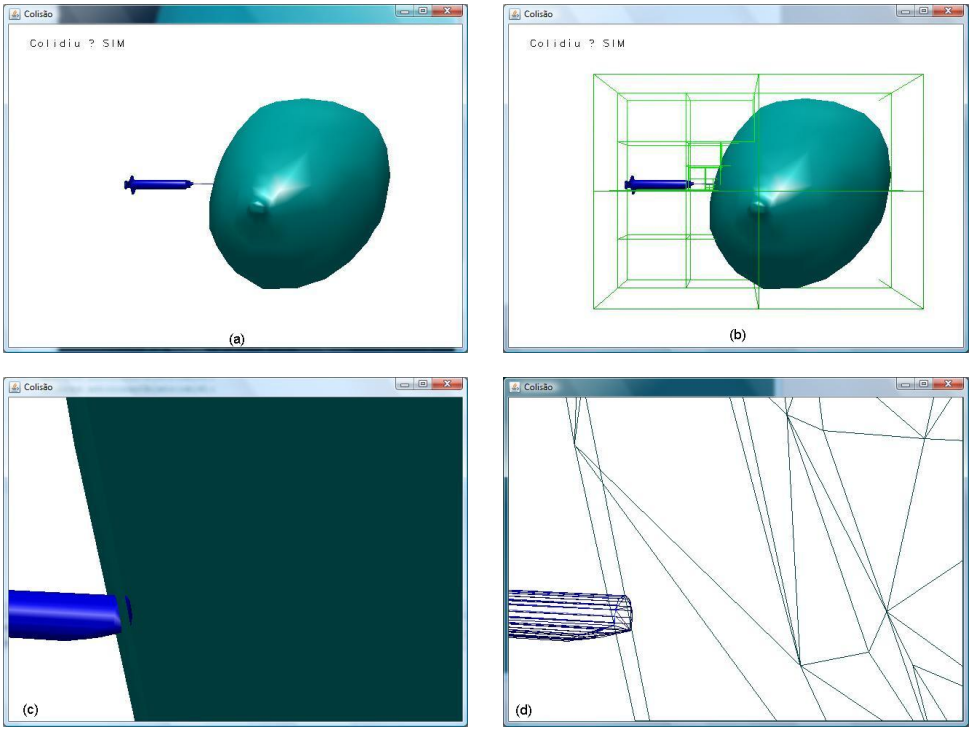

Figura 6: Colisão entre objetos. (a) momento da colisão; (b) sobreposição da *octree*; (c) detalhe da colisão; (d) detalhe da colisão com malhas poligonais visíveis.

4.3 Deformação dos Objetos

Nesta seção são mostrados os testes realizados no ambiente desenvolvido com a adição do método de deformação. O cálculo de deformação é feito apenas após a colisão. Por esse motivo são mostrados apenas os resultados do cálculo da deformação, uma vez que o desempenho da fase anterior a este cálculo não é alterado.

Os valores da constante da mola (K), constante de amortecimento (D), massa (M) e força (F), definidos na seção 3, não foram alterados durante os testes. Os valores, determina-

dos experimentalmente, foram $K = 0.3$, $M = 300.0$, $D = 0.7f$ e $F = (4.0f, 0.0f, 0.0f)$, sendo que F é considerada para os três eixos (x, y e z).

Os objetos *seringa* e *mama* foram novamente utilizados nos testes. A razão de aspecto dos objetos na cena foi alterada para destacar melhor a deformação dos objetos. A Figura 7(a) ilustra os objetos *seringa* e *mama* antes de se colidirem, tal que os objetos estão envolvidos na representação *octree*. Para facilitar a visualização da cena, a Figura 7(b) mostra a mesma imagem anterior com destaque para as malhas poligonais.

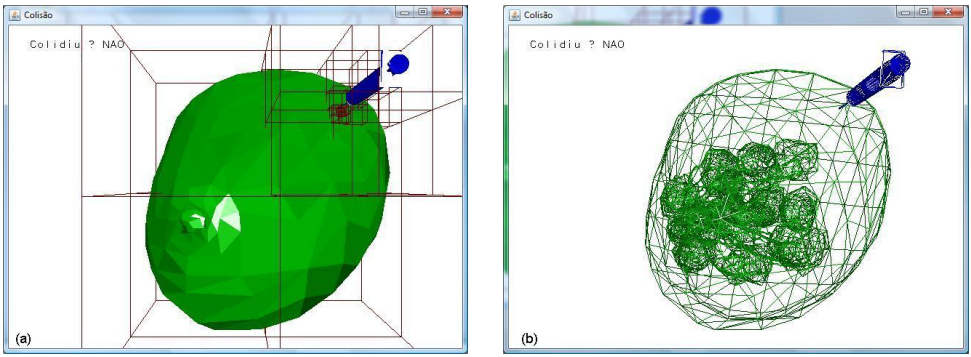

Figura 7: Colisão de objetos. (a) antes da colisão; (b) antes da colisão com malhas poligonais visíveis.

As Figuras 8(a) e 8(b) mostram uma sequência de deformação das faces do objeto *mama* após a colisão. As médias de renderização obtidas durante a deformação nas plataformas Windows e Linux foram de 17 FPS e 16 FPS, respectivamente.

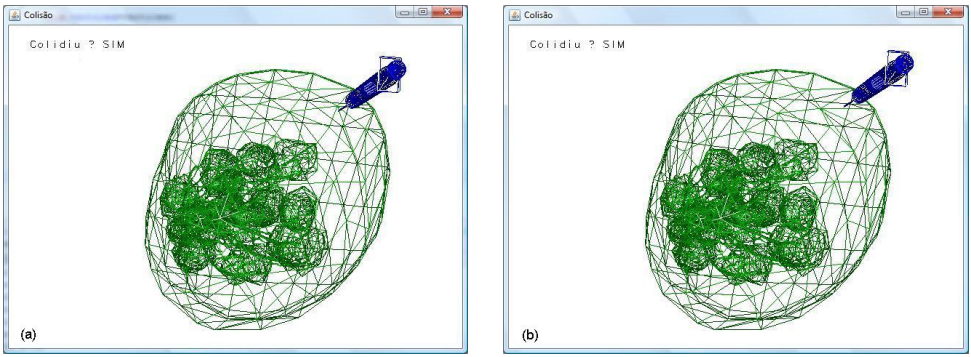

Figura 8: Deformação de face do objeto *mama*.

4.4 Análise dos Resultados

A ideia do AV para treinamento médico é permitir realismo, de forma que as interações forneçam respostas em tempo real e com precisão em relação aos procedimentos de colisão. A precisão em relação à visualização da colisão é medida por meio da distância máxima entre os objetos requerida para detectar a interpenetração entre eles, definida no início dos testes

Em relação ao tempo de resposta, este pode ser medido pela taxa de FPS. Por isso, os resultados apresentados a seguir têm o objetivo de mostrar que esses dois parâmetros foram atendidos para ambas as plataformas de teste.

A partir dos experimentos realizados, pode-se efetuar uma comparação entre os resultados obtidos pelo protótipo para os objetos nas plataformas Windows e Linux. Os valores médios para a taxa de renderização são mostrados na Tabela 1.

Média de FPS	**Colisão**	**Deformação**
Windows	24	17
Linux	23	16

Tabela 1: Comparação entre valores de FPS para os objetos nas plataformas Windows e Linux.

Como o sistema visual humano é capaz de perceber animações em tempo real a taxas próximas de 24 FPS, o ambiente permitiu que o usuário interagisse satisfatoriamente com os objetos (notar que a movimentação dos objetos possui taxas médias em torno de 17 FPS, ou seja, próximo de 24 FPS). Para a deformação foram coletadas as posições iniciais das faces envolvidas na colisão e a posição final das mesmas, mostrando, assim, o grau de deformação.

A Figura 9 ilustra uma sequência de deformação do objeto *mama*. A Figura 9(a) apresenta o objeto em sua forma inicial, a Figura 9(b) mostra o momento da colisão entre os objetos *seringa* e *mama* e o início da deformação, enquanto a Figura 9(c) exibe o objeto após a deformação.

A Tabela 2 mostra, na primeira coluna, o índice correspondente ao vértice do objeto *mama* que foi modificado. As três próximas colunas apresentam a posição (x, y, z) desse vértice em seu estado original e as três últimas colunas exibem a posição final (x, y, z) do vértice após a deformação. Em nossa abordagem, apenas vértices pertencentes a três camadas de adjacência foram considerados em torno do vértice de colisão. A Figura 4 apresentou a formação de camadas de adjacência em uma malha poligonal.

Para confirmação dos resultados em relação à precisão e ao tempo de resposta, outros objetos foram utilizados nos experimentos. A Figura 10(a) ilustra o início de uma sequência de deformação do objeto *nádegas*, mostrando o mesmo com seus vértices originais. A Figura 10(b) mostra o momento da colisão entre os objetos. A Figura 10(c) exibe o final da

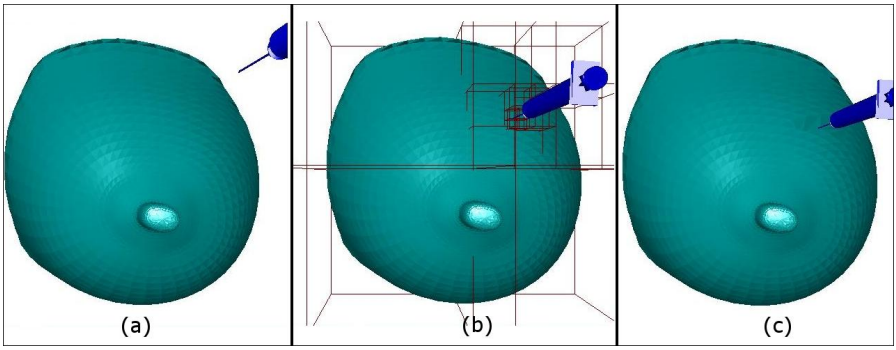

Figura 9: Sequência de deformação do objeto *mama*.

Índice do Vértice	**Início - Sem deformação (Coordenadas x, y, z)**			**Final - Com deformação (Coordenadas x, y, z)**		
1033	2.790618	7.593750	8.822466	2.3826656	6.484224	7.376596
303	0.649441	7.817082	9.075597	0.5543608	6.674912	7.749357
1473	2.781990	9.013126	7.983518	2.3754556	7.696336	6.816769

Tabela 2: Posições dos vértices antes e após a deformação do objeto *mama*.

deformação ocorrida pelo contato entre os objetos *seringa* e *nádegas*. A Tabela 3 mostra as posições iniciais e finais dos vértices deformados.

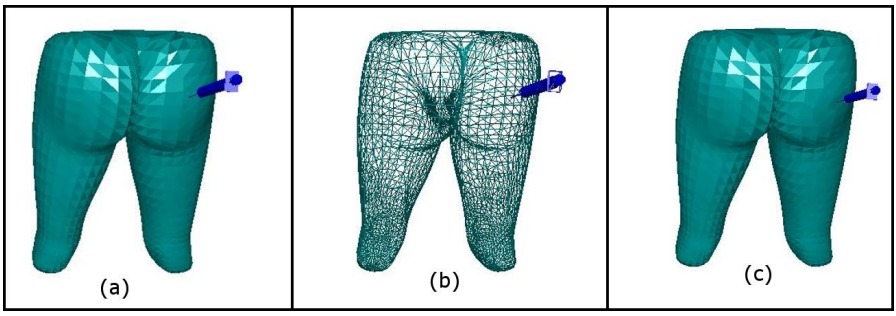

Figura 10: Sequência de deformação do objeto *nádegas*.

A Figura 11 ilustra uma sequência de deformação do objeto *perna*. A Figura 11(a) apresenta o objeto com seus vértices originais, a Figura 11(b) mostra o momento da colisão entre os objetos *seringa* e *perna* e, finalmente, a Figura 11 mostra o objeto *perna* após a

Índice do Vértice	**Início - Sem deformação (Coordenadas x, y, z)**			**Final - Com deformação (Coordenadas x, y, z)**		
846	5.399398	5.716644	8.164954	4.610506	4.881189	6.815141
195	6.668903	6.022062	7.506981	5.694274	5.142079	6.4099874
426	5.476997	4.670556	7.786890	4.676944	3.988847	6.6491313

Tabela 3: Posições dos vértices antes e após a deformação do objeto *nádegas*.

deformação. A Tabela 4 mostra as posições iniciais e finais dos vértices deformados.

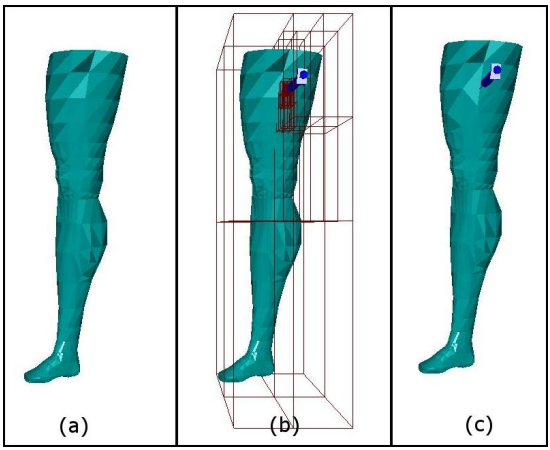

Figura 11: Sequência de deformação do objeto *perna*.

Índice do Vértice	**Início - Sem deformação (Coordenadas x, y, z)**			**Final - Com deformação (Coordenadas x, y, z)**		
389	-0.3521803	11.213284	11.378992	-0.3007779	9.573436	9.559213
375	-1.7953256	9.217928	10.524809	-1.5335817	7.872452	8.987516
362	-1.5345905	7.555994	9.426216	-1.5238687	7.503209	9.360363

Tabela 4: Posições dos vértices antes e após a deformação do objeto *perna*.

5 Conclusões e Trabalhos Futuros

Este trabalho apresentou o desenvolvimento de um protótipo de ambiente virtual interativo para treinamento médico. Métodos de detecção de colisão e de deformação dos objetos

foram incorporados ao ambiente.

Construir ferramentas para treinamento utilizando técnicas de Realidade Virtual exige alguns requisitos inerentes ao domínio da aplicação. Embora outros ambientes e bibliotecas para treinamento médico estejam disponíveis na literatura, o *framework ViMeT* para o qual as técnicas aqui apresentadas foram construídas, visa permitir a geração de aplicações para simular exames de biópsia. Essas simulações exigem precisão na detecção de colisão, a fim de permitir realismo durante o treinamento e tempos de respostas que evitem que o usuário perceba atrasos e comprometa a sensação de reação imediata às suas ações. Esses dois requisitos foram objeto do presente estudo. O mérito científico do trabalho está em propor um método híbrido que permita atingir os requisitos citados.

Estruturas em classes foram projetadas para tornar mais simples a inclusão de novas funcionalidades ao *framework* ViMeT. O desenvolvimento utilizou pacotes de código aberto e multiplataforma com a linguagem de programação Java.

A subdivisão hierárquica do espaço baseada em *octrees* foi utilizada na implementação do método de detecção de colisão dos objetos. O método se mostrou preciso e com adequadas taxas de renderização, permitindo uma simulação com realismo.

O uso da deformação empregando a técnica massa-mola permitiu a simulação de alteração na forma dos objetos que se colidem modificando a posição dos vértices de malhas poligonais, o que proporcionou maior realismo à cena.

Embora o trabalho possa ser melhorado em vários aspectos, conforme propostas de trabalhos futuros citadas a seguir, os experimentos demonstraram que a abordagem atingiu os principais objetivos propostos, ou seja, a criação de um protótipo de ambiente virtual para treinamento médico com alto nível de precisão, bom desempenho e realismo.

Algumas diretrizes de pesquisa podem ser sugeridas a partir dos resultados obtidos neste trabalho. Com a integração das classes e métodos desenvolvidos no *framework* ViMeT, o trabalho poderá trazer ainda mais benefícios a aplicações de treinamento médico. Assim, de uma forma rápida e prática, será possível gerar aplicações para simular diversos tipos de exames de biópsia, simplesmente alterando-se os objetos envolvidos e selecionando-se parâmetros adequados para os processos de colisão e deformação.

Embora os quesitos de precisão e desempenho tenham sido adequadamente alcançados, novos testes devem ser realizados para otimizar os métodos por meio da modificação dos parâmetros utilizados. Em particular, sugere-se um estudo mais detalhado dos parâmetros da deformação massa-mola, buscando a obtenção de maior realismo na representação de tecidos de órgãos humanos.

A taxa de renderização obtida durante a movimentação dos objetos ainda não é plenamente adequada, o que pode ser melhorada a partir de otimizações nos algoritmos de detecção

de colisão e deformação de objetos, bem como aceleração da execução por placas gráficas.

Finalmente, outra proposta de trabalho futuro é a incorporação de equipamentos não convencionais, melhorando a interação entre o usuário e o ambiente. Sugere-se inicialmente a inclusão de dispositivo háptico para possibilitar a percepção de retorno de força ao usuário quando ocorrer uma colisão. Também está prevista a utilização de uma luva de dados para simular o ato de segurar o objeto virtual que representa o órgão humano, dessa forma, inserindo mais realismo em relação ao procedimento que é executado no treinamento médico.

6 Referências

[1] ANTONIO, F. *Faster Line Segment Intersection*. Academic Press Professional, Inc., San Diego, CA, Estados Unidos, 1992, pp. 199–202.

[2] AUKSTAKALNIS, S. E BLATNER, D. *The Art and Science of Virtual Reality*. Peatchpit Press, Berkeley, CA, Estados Unidos, 1992.

[3] AUTODESK. Autodesk 3ds Max, 2008. `http://usa.autodesk.com/`, acesso em 30 de julho de 2008.

[4] BENTLEY, J. L. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM 18*, 9 (1975), 509–517.

[5] CHOI, K. S., SUN, H., HENG, P. A. E CHENG, J. C. Y. A Scalable Force Propagation Approach for Web-based Deformable Simulation of Soft Tissues. In *Seventh International Conference on 3D Web Technology* (Tempe, AZ, Estados Unidos, 2002), pp. 185–193.

[6] DE BERG, M., VAN KREVELD, M., OVERMARS, M. E SCHWARZKOPF, O. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, New York, NY, Estados Unidos, 2000.

[7] GIBSON, S. F. F. E MIRTICH, B. A Survey of Deformable Modeling in Computer Graphics. Relatório Técnico TR-97, Mitsubishi Electric Research Laboratories, Cambridge, MA, Estados Unidos, 1997.

[8] GOTTSCHALK, S., LIN, M. C. E MANOCHA, D. OBBTree: A Hierarchical Structure for Rapid Interference Detection. In *23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, Estados Unidos, 1996), ACM Press, pp. 171–180.

[9] GUIBAS, L. J., HSU, D. E ZHANG, L. H-Walk: Hierarchical Distance Computation for Moving Convex Bodies. In *Fifteenth Annual Symposium on Computational Geometry* (Miami Beach, FL, Estados Unidos, 1999), pp. 265–273.

[10] HANCOCK, D. Viewpoint: Virtual Reality in Search of Middle Ground. *IEEE Spectrum 32*, 1 (Janeiro 1995), 68.

[11] HEARN, D. E BAKER, M. P. *Computer Graphics, C Version*. Prentice Hall, 1996.

[12] HIROTA, G., MAHESHWARI, R. E LIN, M. C. Fast Volume-Preserving Free Form Deformation using Multi-level Optimization. In *Fifth ACM Symposium on Solid Modeling and Applications* (Ann Arbor, MI, Estados Unidos, 1999), pp. 234–245.

[13] HUBBARD, P. M. Collision Detection for Interactive Graphics Applications. *IEEE Transactions on Visualization and Computer Graphics 1*, 3 (1995), 218–230.

[14] HUTCHINS, M., STEVENSON, D., GUNN, C., KRUMPHOLZ, A., PYMAN, B. E O'LEARY, S. "I think I can see it now!": Evidence of Learning in Video Transcripts of a Collaborative Virtual Reality Surgical Training Trial. In *17th Australia Conference on Computer-Human Interaction* (Narrabundah, Austrália, 2005), Computer-Human Interaction Special Interest Group (CHISIG) of Australia, pp. 1–4.

[15] JOGL.DEV.JAVA.NET. JOGL - User's Guide, Outubro 2008. `https://jogl.dev.java.net/nonav/source/browse/*checkout*/jogl/doc/userguide/index.html?rev=HEAD&contenttype=text/html`.

[16] JOHNSEN, K., RAIJ, A., STEVENS, A., LIND, D. S. E LOK, B. The Validity of a Virtual Human Experience for Interpersonal Skills Education. In *SIGCHI Conference on Human Factors in Computing Systems* (2007), ACM, pp. 1049–1058.

[17] KERA, M. Ambiente Virtual Interativo com Colisão e Deformação de Objetos para Treinamento Médico Utilizando a API Java. Dissertação de Mestrado, Universidade Federal do Paraná, Curitiba-PR, Brasil, Agosto 2008.

[18] KERA, M., BRUNO FILHO, J. W. E PEDRINI, H. Análise Comparativa entre Ferramentas para Aplicações em Realidade Virtual. In *II Workshop de Aplicações de Realidade Virtual* (Recife-PE, Brasil, Novembro 2006), pp. 13–16.

[19] KIRNER, C. E PINHO, M. Uma Introdução à Realidade Virtual. In *Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens* (Campos do Jordão, SP, Brasil, Outubro 1996).

[20] KLOSOWSKI, J. T., HELD, M., MITCHELL, J. S., SOWIZRAL, H. E ZIKAN, K. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics 4*, 1 (1998), 21–36.

[21] LIN, M. C. E CANNY, J. F. Efficient Algorithms for incremental Distance Computation. *IEEE Conference on Robotics and Automation 2* (1991), 1008–1014.

[22] MA, J. E NICKERSON, J. V. Hands-on, Simulated, and Remote Laboratories: A Comparative Literature Review. *ACM Computing Surveys 38*, 3 (2006).

[23] MACHADO, L. E ZUFFO, M. Development and Evaluation of a Simulator of Invasive Procedures in Pediatric Bone Marrow Transplant. *Studies In Health Technology And Informatics 94* (2003), 193–195.

[24] MOORE, P. E MOLLOY, D. A Survey of Computer-Based Deformable Models. In *International Machine Vision and Image Processing Conference* (Maynooth, Irlanda, Setembro 2007), pp. 55–66.

[25] MORIE, J. F. Inspiring the Future: Merging Mass Communication, Art, Entertainment and Virtual Environments. *ACM SIGGRAPH Computer Graphics 28*, 2 (1994), 135–138.

[26] NUNES, F., MACHADO, L. E COSTA, R. RV e RA Aplicadas à Saúde. In *Aplicações de Realidade Virtual e Aumentada*, R. Costa and M. Wagner, Eds. Sociedade Brasileira de Computação (SBC), Porto Alegre, RS, Brasil, 2009, pp. 69–89.

[27] OLIVEIRA, A. C. M. T. G. ViMeT - Projeto e Implementação de um Framework para Aplicações de Treinamento Médico usando Realidade Virtual. Dissertação de Mestrado, Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, SP, Brasil, 2007.

[28] OLIVEIRA, A. C. M. T. G. E NUNES, F. L. S. Building an Open Source Framework for Virtual Medical Training. *Journal of Digital Imaging 23*, 6 (2010), 706–720.

[29] O'REILLY. Online Mirror of the Encyclopedia of Graphics File Formats. `http://www.fileformat.info/format/wavefrontobj/`.

[30] PONAMGI, M. K., MANOCHA, D. E LIN, M. C. Incremental Algorithms for Collision Detection Between Polygonal Models. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (1997), 51–64.

[31] PROVOT, X. Deformation Constraints in a Mass-spring Model to Describe Rigid Cloth Behavior. In *Proceedings of Graphics Interface* (Quebec, Canadá, 1995), pp. 147–154.

[32] SUN MICROSYSTEMS. Tutorials and Code Camps, Java 3D API Tutorial, Outubro 2006. `http://developer.java.sun.com/developer/onlineTraining/java3d`.

[33] SUN MICROSYSTEMS. The Java 3DTM API Specification Version 1.3, Maio 2007. `http://java.sun.com/products/javamedia/3D/forDevelopers/J3D_1_3_API/j3dguide`.

[34] THIBAULT, W. C. E NAYLOR, B. F. Set Operations on Polyhedra using Binary Space Partitioning Trees. *ACM SIGGRAPH Computer Graphics 21*, 4 (1987), 153–162.

[35] VALERIO NETTO, A., MACHADO, L. E OLIVEIRA, M. Realidade Virtual - Definições, Dispositivos e Aplicações. *Revista Eletrônica de Iniciação Científica 2*, 1 (2002).

[36] VAN DEN BERGEN, G. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools 2*, 4 (1997), 1–13.

[37] VON SCHWEBER, L. E VON SCHWEBER, E. Cover Story: Realidade Virtual. *PC Magazine Brasil 5*, 6 (1995), 50–73.