FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
CENTRO UNIVERSITÁRIO "EURÍPIDES DE MARÍLIA" – UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARCELLO KERA

DETECÇÃO DE COLISÃO UTILIZANDO HIERARQUIAS EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

MARÍLIA
2005

MARCELLO KERA

DETECÇÃO DE COLISÃO UTILIZANDO HIERARQUIAS EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

Monografia apresentada ao curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília, mantido pela Fundação Eurípides Soares da Rocha, para a obtenção do título de Bacharel em Ciência da Computação.

Orientadora: Profa. Dra. Fátima L. S. Nunes

MARÍLIA
2005

Kera, Marcello
Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico / Marcello Kera; orientador: Fátima L. S. Nunes
Marília, SP: [s.n.], 2005.
92 f.

Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha

1. Detecção de Colisão 2. Java3D 3. Realidade Virtual

CDD: 006

Dedico este trabalho a Aquele que me apoiou e me salvou nos momentos de dor, sofrimento e desespero. É em quem eu me apoio e sempre vou me apoiar.

AGRADECIMENTOS

Agradeço:

A Deus, pela ajuda sobrenatural nos momentos desesperadores da vida.

À minha Mãe, Dona Marilú, por nunca ter desistido de mim e ter me bancado durante toda a minha jornada.

À Fátima, minha orientadora, pela enorme confiança na minha capacidade.

Aos meus amigos e irmãos de coração: Vasco, Sys (Gustavo), Gzus (Rodrigo), Carlim (Carlos), Guilherme e a galera da sala que sempre esteve presente.

Por fim, a todos aqueles que direta ou indiretamente me ajudaram com esse projeto.

KERA, Marcello. Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico. 2005 92 f. Monografia (Bacharelado em Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005

RESUMO

Precisão e desempenho em tempo real são requisitos fundamentais para aplicações de Realidade Virtual em treinamento médico. Nessas aplicações, a detecção de colisão é de extrema importância para simular com realismo os procedimentos do usuário. Este trabalho apresenta uma análise dos algoritmos de detecção de colisão nativos da API Java3D e a construção de um novo método que proporciona detecção com precisão, utilizando uma ferramenta para simular o exame de punção mamária como estudo de caso.

Palavras-chave: Detecção de Colisão, Java3D, Realidade Virtual.

KERA, Marcello. Detecção de colisão utilizando hierarquias em ferramentas de realidade virtual para treinamento médico. 2005 92 f. Monografia (Bacharelado em Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005

ABSTRACT

Precision and performance in real time are main requirements for virtual reality tools for medical training. In such applications the collision detection is extremely important to simulate the user's proceedures. This work presents an aanalysis of collision detection algorithms that are native on the Java3D API and the construction of a new method that perform the detection with precision, using a tool to simulate breast biopsy as a case study.

Keywords: Collision Detection, Java3D, Virtual Reality.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS

AABB - axis-aligned bounding boxes

API – *Application Programming Interface*

ARV – Ambiente de Realidade Virtual

AV – Ambiente Virtual

BB – *BoundingBox*

BS – *BoundingSphere*

CoU – *Configured Universe*

DCF - Detecção de colisão por faces

FPS – Frames por Segundo

HDM – *Head Mounted Display*

]J3D – Java3D

K-Dops - D*iscrete orientation polytopes*

OBB - O*riented Bounding Box*

QuOSPO - *Quantized Orientation Slabs with Primary Orientations*

RV – Realidade Virtual

SU – *Simple Universe*

URL - *Universal Resource Locator*

VU – *Virtual Universe*

VRML - *Virtual Reality Modeling Language*

SUMÁRIO

INTRODUÇÃO

A Realidade Virtual (RV) pode fornecer mecanismos mais adequados para construção de ferramentas que exigem interação avançada, pois, a RV é uma forma das pessoas visualizarem, manipularem e interagirem com computadores e dados extremamente complexos (AUKSTAKALNIS e BLATNER, 1992).

A detecção de colisão é um dos itens mais complexos e dependentes das informações de interação monitoradas em um Ambiente Virtual (AV), permitindo responder às interações entre objetos no mundo virtual, fator importante para obtenção do realismo. Este requisito exige programas específicos para gerenciar as informações de interação, envolvendo rotinas de controle e computação gráfica (MACHADO et al, 2002).

Na etapa da detecção de colisão, a simulação deve emitir uma resposta automática ao encontro entre objetos, como a deformação, um salto, restrição ao movimento ou mesmo produzir forças e vibrações (BURDEA, 1997).

O problema da detecção de colisão entre objetos é fundamental em qualquer simulação do mundo físico, sendo estudado por diversas comunidades, incluindo a robótica, a computação gráfica e a geometria computacional. Diante disso, existem diversos métodos e algoritmos para verificar contato entre objetos. Como a execução de algoritmos mais refinados pode causar um custo de processamento maior, soluções mais simples, como a abordagem hierárquica (O'SULLIVAN et al, 2001), normalmente são adotadas.

Objetivos

O objetivo deste trabalho é realizar uma avaliação de algoritmos de detecção de colisão presentes na API Java3D (SUN, 2005) e implementar um algoritmo que pudesse melhorar o desempenho e a precisão de tais métodos. Para definir de forma mais efetiva as comparações

foram focalizadas as aplicações para treinamento médico, que exigem níveis altos de desempenho e precisão nos algoritmos de detecção de colisão.

Para atingir o objetivo, foram inicialmente estudados vários algoritmos identificados na literatura, focando o desenvolvimento de ferramentas de RV para treinamento médico. A partir do levantamento de literatura realizado, foi escolhido um algoritmo de abordagem hierárquica para favorecer as almejadas características de precisão e desempenho.

Estrutura do trabalho

Além desta introdução, este trabalho está organizado da seguinte forma:

Capítulo 2 – Realidade Virtual e a medicina: este capítulo apresenta uma breve introdução aos conceitos de RV, a RV na medicina e algumas aplicações de RV de simulação e treinamento médico.

Capítulo 3 - Ambientes Virtuais e Java3D: descreve conceitos de ambientes virtuais.

Capítulo 4 – Detecção de Colisão: neste capítulo é abordado o conceito de detecção de colisão, através de uma revisão bibliográfica.

Capítulo 5 – Implementação: aborda o desenvolvimento do projeto descrevendo as particularidades da implementação.

Capítulo 6 – Resultados: monstra os resultados obtidos com o projeto desenvolvido.

Capítulo 7 – Conclusões: apresenta as considerações finais sobre o projeto e a indicação de trabalhos futuros.

Ao final, são disponibilizadas as referências bibliográficas citadas ao longo do texto desta monografia.

CAPÍTULO 1 - REALIDADE VIRTUAL E A MEDICINA

Algumas ferramentas de RV estão sendo desenvolvidas para o treinamento médico em vários centros de pesquisa do mundo, com o objetivo de auxiliar profissionais da saúde em procedimentos usuais. As aplicações voltadas para visualização científica, treinamento e simulação de procedimentos médicos têm sido o foco da maioria dos projetos.

Neste capítulo serão apresentados os principais conceitos de RV, os aspectos da RV aplicada à medicina e uma variedade de projetos de simulação e treinamento médico.

1.1 Conceitos de Realidade Virtual

O termo RV possui diversas definições, tanto na área acadêmica quanto na área comercial. De uma forma direta pode ser definida como uma interface natural e poderosa de interação homem-máquina, por permitir ao usuário interação, navegação e imersão num ambiente tridimensional sintético, gerado por computador, através de canais multisensoriais, tais como visão, audição, tato etc. (KIRNER, 1996).

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, dispositivos hápticos[1] podem ser utilizados para a visualização desse mundo e para que o usuário tenha uma sensação de estar em um mundo real. (NETTO et al, 2002).

Latta e Oberg (1994) afirmam que a RV envolve a criação e experimentação de

[1] Disposiivo háptico - Pode ser chamado de dispositivos de reação tátil, esses dispositivos estimulam sensações como tato, tensão muscular e temperatura (GRADECKI, 1995).

ambientes, tendo como objetivo colocar o usuário em um ambiente que não pode ser encontrado trivialmente no mundo real estabelecendo uma ligação entre ambos.

NETTO et al (2002) lembram que a partir do momento que o usuário pode interagir com o mundo virtual, por meios de dispositivos de entrada e saída, esse mundo virtual passa a ser um Ambiente Virtual (AV) ou um Ambiente de RV (ARV), mas ainda diz que três aspectos têm que ser considerados para esse ambiente se tornar um AV ou ARV: imersão, interação e envolvimento.

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 utilizadas, enquanto a não imersiva apenas utiliza 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 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 no AV, ou ativo, como participar de um jogo.

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, e a inclusão da descrição funcional dos objetos (NETTO et al*,* 2002).

1.2 Aspectos da Realidade Virtual na medicina

A variedade de aplicações em RV na Medicina são muito amplas, envolvendo desde

estudos anatômicos como a visualização de um atlas virtual, até aplicações de simulação e treinamento médico.

A simulação cirúrgica baseada em RV tem sido fonte de pesquisa em todo o mundo, objetivando substituir no futuro próximo, os atuais métodos de treinamento e planejamento de procedimentos médicos, podendo representar uma alternativa para um custo mais efetivo e eficiente (MACHADO et al, 2001; KÜHNAPFEL et al, 2000).

Simuladores cirúrgicos em RV estão sendo desenvolvidos para fornecer informações detalhadas sobre os tecidos, as ferramentas e as ações inerentes a certos procedimentos (TSAI et al, 2000).

Segundo Riquelme (2005) usando técnicas de interação baseadas em RV, futuramente cirurgiões poderão adquirir habilidades básicas para manipularem instrumentos utilizados em procedimentos médicos, aprender novos procedimentos e aumentar o nível de habilidade antes de efetuar o procedimento em um paciente real.

É importante o uso de modelos anatômicos com propriedades físicas que permitam a percepção, visual ou por meio de dispositivos hápticos, a forma do organismo, sendo importante à escolha de uma ferramenta de manipulação adequada para o procedimento, e o desenvolvimento de rotinas gráficas e de resposta tátil para a visualização e interação do usuário (MACHADO, 2003).

Como mencionado, existem diversos projetos de simulação e treinamento médico citados na literatura sendo que vários ainda estão em desenvolvimento. A seguir, são apresentados alguns desses trabalhos.

Burdea et al (1999) apresentam um simulador baseado em RV para diagnóstico de câncer de próstata. O diagnóstico virtual é realizado por meio de um anel que fornece retorno tátil para o usuário (Figura 1), tendo como retorno um de quatro diferentes casos: próstata

normal, próstata aumentada, tumor em estado inicial (Figura 2) e tumor em estado avançado (Figura 3). Uma funcionalidade muito importante que a aplicação fornece é permitir a gravação de exames previamente simulados e a reprodução desses para uma posterior apresentação.

Uma outra aplicação é um simulador para treinamento de acupuntura lombar apresentado por Gorman et al (2000), com o objetivo de aumentar a habilidade dos estudantes evitando erros que poderiam causar dores desnecessárias e ferimentos em pacientes reais.

Este simulador consiste de uma agulha para acupuntura lombar passada pelo dorso de um manequim humano e unida a um dispositivo háptico (Figura 4). O dispositivo háptico permite ao estudante sentir forças resistentes sobre uma trajetória correta ou incorreta. Por exemplo, em uma colisão com osso o simulador também permite a gravação das ações do usuário. A Figura 5 monstra como é visualizada a penetração da agulha no AV.

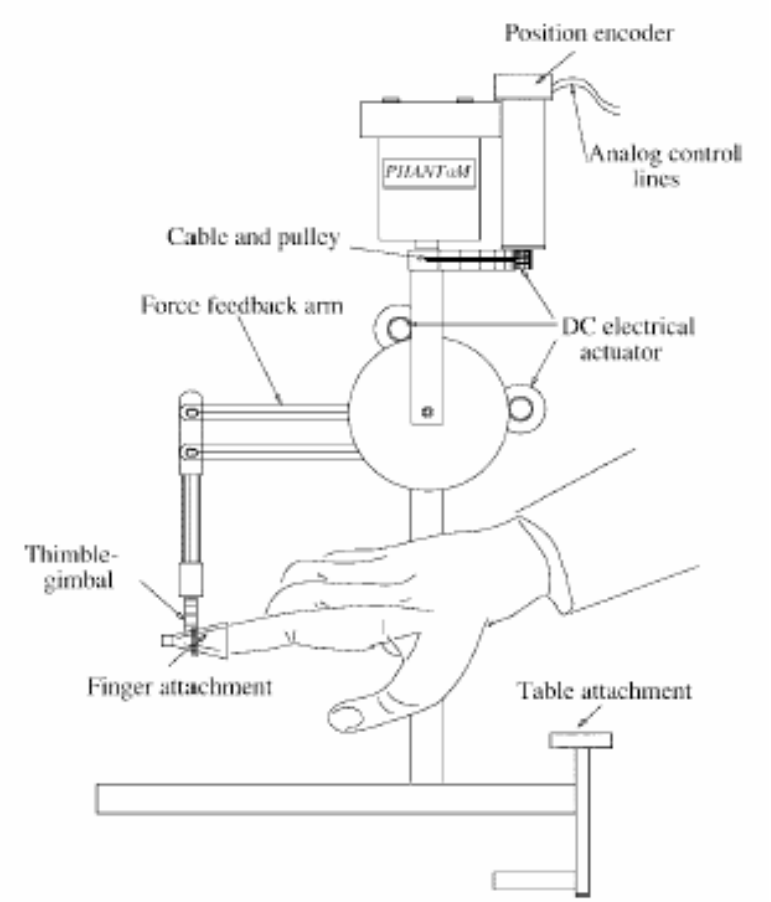


Figura 1: Configuração do dispositivo háptico *Phantom* (BURDEA et al, 1999).

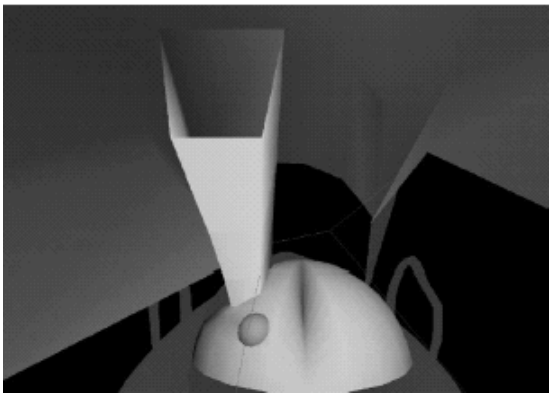


Figura 2: Visão interior de uma próstata com um tumor em estado inicial (BURDEA et al, 1999).

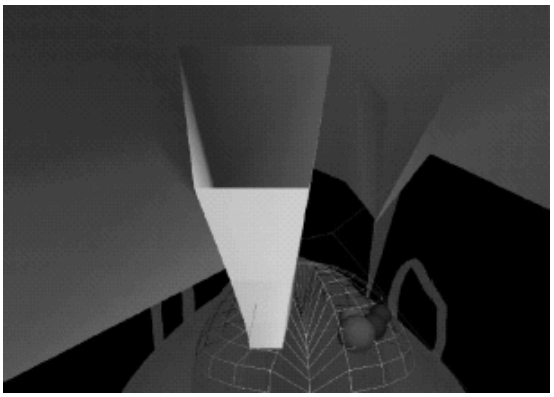


Figura 3: Visão interior de uma próstata transparente com um tumor em estado avançado (BURDEA et al, 1999).

Figura 4: Simulador para treinamento em acupuntura lombar (GORMAN et al, 2000).

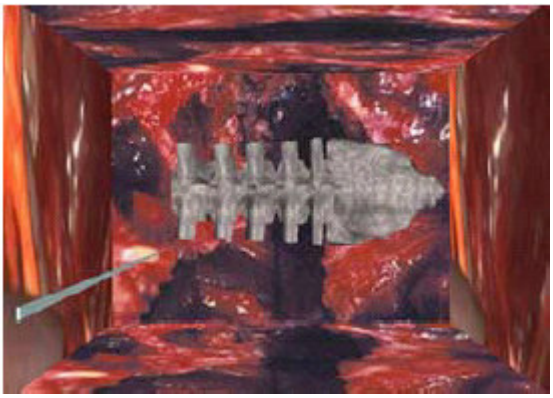


Figura 5: Modelo 3D de uma espinha em VRML (GORMAN et al, 2000).

O treinamento de laparoscopia, que consiste em um procedimento cirúrgico para exame da cavidade abdominal e pélvica, é um objeto de estudo muito importante para RV, pois todo o procedimento é realizado com uma câmera dentro do paciente, e as imagens fornecidas por essa

câmera é a única fonte de informação para o cirurgião necessitando assim um treinamento grande para a cirurgia. Alguns simuladores em RV de treinamento para laparoscopia já foram desenvolvidos, como os apresentados por Tendick et al (2000), Haluck et al (2001), Webster et al *(*2003) e o sistema LapSim (LAPSIM, 2005).

As Figuras 6 e 7 mostram a aplicação de Tendick et al (2000) sendo a primeira à visualização do AV e a segunda mostrando a utilização do dispositivo háptico. As Figuras 8 e 9 mostram a aplicação de Webster et al (2003) e o dispositivo háptico utilizado respectivamente, sendo que esse pode dar ao usuário uma resposta háptica de alto nível. O sistema LapSim (LAPSIM, 2005) é dividido em três módulos, sendo um para habilidades básicas como corte e sutura (Figura 10), um para o procedimento de dissecação (Figura 11) e outro, mais recente, para cirurgias ginecológicas (Figura 12).

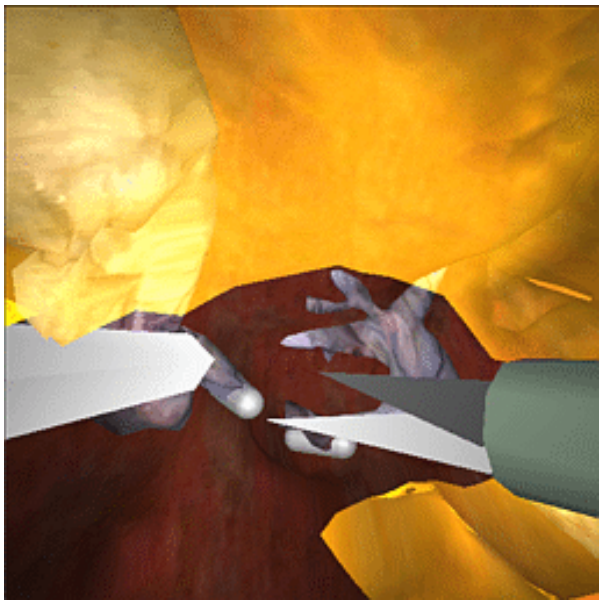


Figura 6: Ambiente simulado para cirurgia laparoscópica (TENDICK et al, 2000).

Figura 7: Interface Háptica (TENDICK et al, 2000).

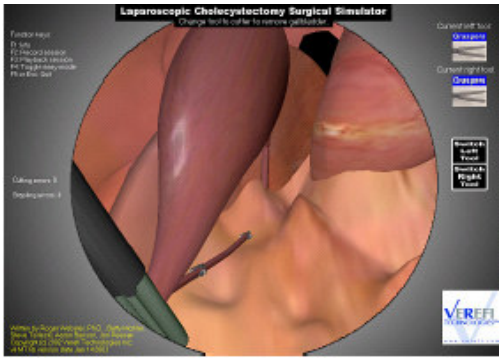


Figura 8: Simulação de cirurgia laparoscópica (WEBSTER et al, 2003).

Figura 9: Estação para simulação de cirurgia laparoscópica (IMMERSION MEDICAL, 2003).

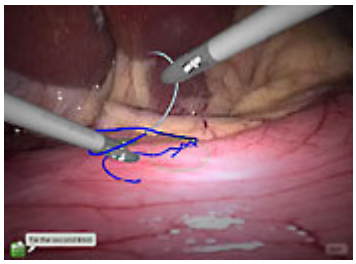


Figura 10: Sutura no módulo BasicSkills (LAPSIM SYSTEM, 2003).

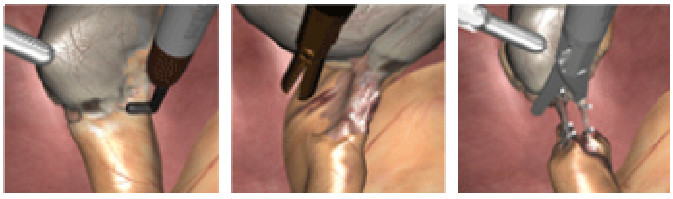


Figura 11: Dissecação no módulo Dissection (LAPSIM SYSTEM, 2003).

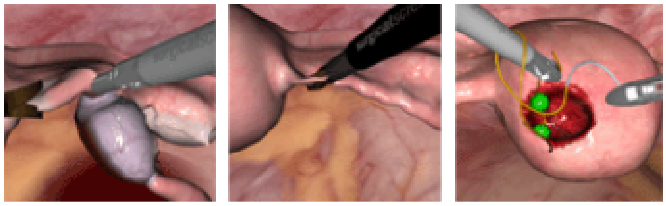


Figura 12: Cirurgia Ginecológica no módulo Gyn (LAPSIM SYSTEM, 2003).

Outros temas abordados para o desenvolvimento de simuladores em RV são o treinamento de cirurgias ortopédicas como descrito em Sourina et al *(*2000) e cirurgias pediátricas em Machado (2003).

Em um treinamento de cirurgia ortopédica, normalmente ossos de plástico (Figura 13) são utilizados para simular a reparação desse osso, usando ferramentas cirúrgicas e implantes, mas todo esse procedimento tem um custo muito elevado e nem todos modelos dos tipos de ossos são encontrados. Por isso foi desenvolvido o simulador *Virtual Bone-setter* (SOURINA et al, 2000), para diminuir os custos do treinamento e para que esse pudesse ser feito em qualquer parte do corpo.(Figura 14).

Figura 13: Típico osso plástico fraturado (SOURINA et al, 2000).

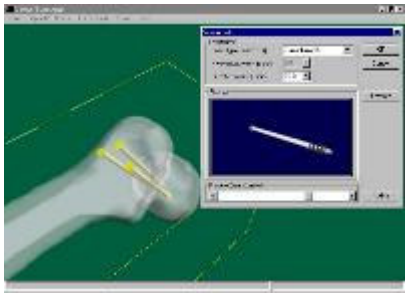


Figura 14: Fixação de uma fratura virtual (SOURINA et al, 2000).

Machado (2003) descreve um simulador de coleta de medula óssea para treinamento de oncologia pediátrica (Figura 15). Um dos motivos do projeto, além da não utilização de animais como cobaias, foi permitir maior usabilidade e disponibilidade no treinamento.

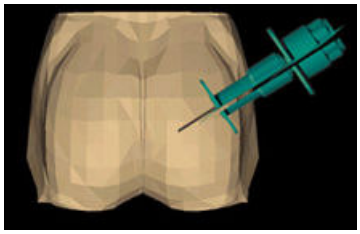


Figura 15: "Agulha virtual" do simulador de coleta de medula óssea (MACHADO, 2003).

Nesse capítulo foi possivel notar que existem inumeras aplicações de RV voltadas para simulação e treinamento médico e que o realismo dessas aplicações são de extrema importância. O desenvolvimento de um algoritmo de detecção de colisão que traga um aumento de realismo para essas aplicações, constituindo o escopo do presente projeto.

CAPÍTULO 2 - AMBIENTES VIRTUAIS EM JAVA3D

Morie (1994) define um Ambiente Virtual (AV) como um ambiente tridimensional – ou espaço imaginário – gerado por computador. Em geral, os ambientes virtuais visam a reproduzir situações do mundo real ou próximas daquelas percebidas no mundo real, com a finalidade de permitir simulação, treinamento, visualização ou outro tipo de atividade por meio de técnicas de RV. Normalmente os ambientes virtuais construídos para aplicações de simulação e treinamento são compostos por dois ou mais objetos que podem se movimentar a partir da interação do usuário. Para propiciar a execução dessas atividades, os AVs normalmente devem prever características inerentes ao mundo real como movimentos, colisões e deformações, a fim imprimir maior realismo às aplicações.

Há várias linguagens para a implementação de AVs. Uma delas é a linguagem Java que juntamente com a API Java3D (J3D) possibilita a criação de um AV. A seguir uma breve explanação sobre a API J3D.

2.1 Java3D

A API J3D consiste em uma hierarquia de classes Java que serve como interface para o desenvolvimento de sistemas gráficos tridimensionais. Possui construtores de alto nível que permitem a criação e manipulação de objetos geométricos, especificados em um universo virtual. Também possibilita a criação de universos virtuais com uma grande flexibilidade, nos quais as cenas são representadas através de grafos, e os detalhes de sua visualização são gerenciados automaticamente. (SUN, 2005)

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 (Figura 16), podendo assim ser tratados tanto individualmente quanto em grupo (BICHO et al, 2002). Os grafos de cena são responsáveis pela especificação do conteúdo do universo virtual e pela forma como este é visualizado.

Um grafo de cena é uma estrutura de dados que descreve uma cena tridimensional na forma de nós. Os nós guardam todas as propriedades dos objetos presentes numa cena, como geometria, cor, transparência e textura. As ligações 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 (SUN, 2005).

J3D foi construída com o objetivo de criar uma API que fosse independente de plataforma, semelhante a *Virtual Reality Modeling Language* (VRML). Atualmente, a J3D consiste em uma API baseada nas bibliotecas gráficas OpenGL e DirectX e os programas podem ser escritos como aplicação, *applet*, ou ambas. Nos últimos anos, várias aplicações foram desenvolvidas usando J3D, tais como jogos, comércio eletrônico, visualização de dados e elaboração de interfaces.

DirectX é uma biblioteca desenvolvida pela Microsoft para PCs baseados no Sistema Operacional Windows, que permite aos desenvolvedores acessar recursos avançados de *hardware* sem a necessidade de escrever códigos específicos para esse fim (MICROSOFT CORPORATION, 2004). Assim, a J3D torna-se uma biblioteca de mais alto nível de abstração quanto à programação gráfica quando comparada a OpenGL ou DirectX.

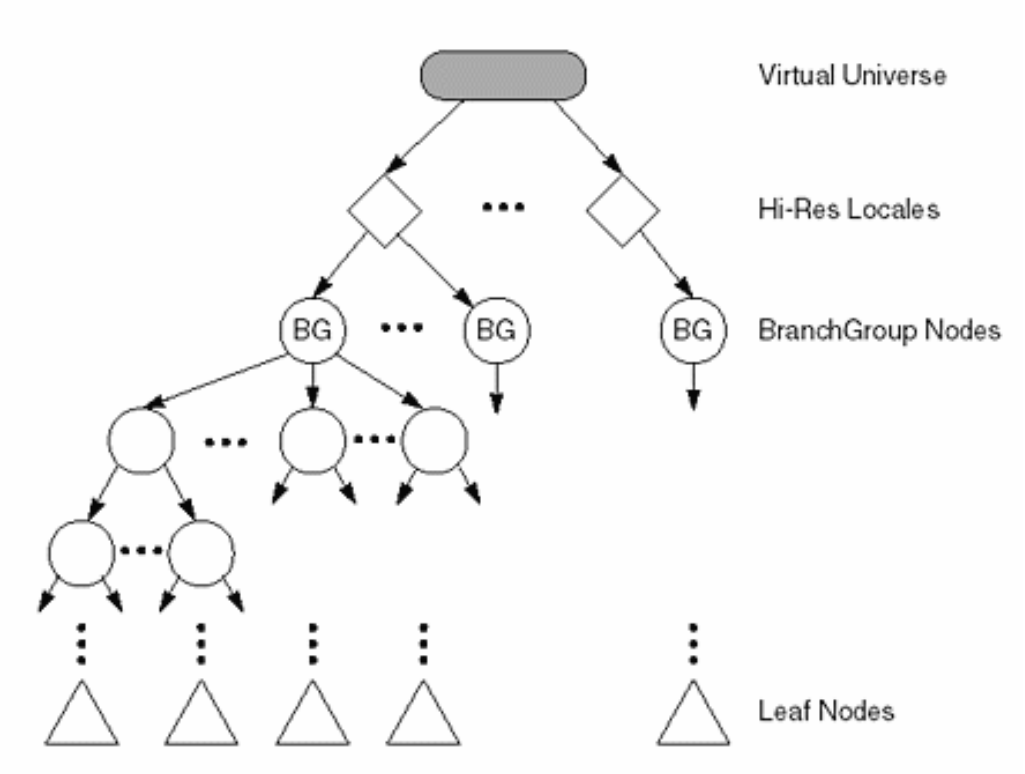


Figura 16: Estrutura Hierárquica do Grafo de Cena em J3D (SUN, 2005).

Na J3D existem três meios de se implementar um AV, que são chamados de Universo: o *SimpleUniverse*, o *VirtualUniverse*, e o *ConfiguredUniverse*, detalhados a seguir.

A classe *SimpleUniverse* permite que se crie facilmente um ambiente mínimo de usuário para um programa J3D ser executado. Cria todos os objetos necessários na parte *View* do grafo de cena. Especificamente, esta classe cria um nó *Locale*, um único nó do tipo *ViewingPlatform*, e um objeto *Viewer* (ambos com valores padrões). Para muitas aplicações básicas de J3D a *SimpleUniverse* provê todas as funcionalidades necessárias às aplicações. Aplicações mais sofisticadas podem precisar de mais controles para adquirir funcionalidades extras. (SUN, 2005)

O *VirtualUniverse* consiste em uma lista de objetos *Locale* sendo que cada um deles contém uma coleção de nós de grafo de cenas existentes no universo. Uma aplicação ou *applet* pode ter mais que um *VirtualUniverse*, mas muitas aplicações precisam somente de um.

Universos virtuais são entidades separadas sendo que nenhum nó pode existir em mais de um universo virtual ao mesmo tempo, assim como, os objetos em um universo virtual não são visíveis dentro, nem interagem com objetos dentro de qualquer outro universo virtual. Um objeto do tipo *VirtualUniverse* define métodos para enumerar seu Locale e os remover do universo virtual. (SUN, 2005)

A classe *ConfiguredUniverse* cria todos os objetos *View* necessários do grafo de cena. Especificamente, cria um *Locale*, um ou mais *ViewingPlatforms*, e pelo menos um objeto de *Viewer*. O *ConfiguredUniverse* pode montar um ambiente baseado nos conteúdos de um arquivo de configuração. Isto permite que uma aplicação seja executada sem mudança por uma gama larga de configurações, como janelas em PC convencionais, óculos 3D, telas de imersão únicas ou múltiplas telas, ou instalações de RV inclusive em Cave´s e exibições HMD´s que permitem os seis graus de liberdade. (SUN, 2005)

Pelo menos um *Viewer* deve ser provido pela configuração. Se um *ViewingPlatform* não é provido, um padrão será criado e o *Viewer* será anexado a ele. Um arquivo de configuração pode ser especificado diretamente passando a *URL* a um construtor do *ConfiguredUniverse*. Alternativamente, um *ConfigContainer* pode ser criado primeiro a partir de um arquivo de configuração e, então, passar a um construtor do *ConfiguredUniverse* apropriado. Se um arquivo de configuração ou *container* não for criado, o *ConfiguredUniverse* cria um ambiente *viewing* padrão da mesma forma como o *SimpleUniverse*. Se forem informados um ou mais objetos *Canvas3D*, ele os usará em vez de criar o novo. Todos os construtores disponíveis para o *SimpleUniverse* também estão disponíveis no *ConfiguredUniverse.* (SUN, 2005)

CAPÍTULO 3 - DETECÇÃO DE COLISÃO

Detectar uma colisão é verificar o momento em que ocorre uma aproximação entre objetos de um ambiente virtual suficientemente pequena a ponto de possibilitar a ocorrência de uma sobreposição entre eles. Tem a finalidade de evitar que objetos virtuais se interpenetrem, fornecendo maior realismo às aplicações de RV. A sua percepção exige a presença de, no mínimo, dois objetos em um ambiente virtual, sendo que pelo menos um deles deve estar em movimento. É um problema complexo, visto que objetos virtuais podem ter formas, tamanhos e movimentos variados, dependendo de suas naturezas e da finalidade da aplicação.

A detecção de colisão é um dos itens mais complexos e dependentes das informações de interação monitoradas em um Ambiente Virtual (AV), permitindo responder às interações entre objetos no mundo virtual, fator importante para obtenção do realismo. Este requisito exige programas específicos para gerenciar as informações de interação, envolvendo rotinas de controle e computação gráfica (MACHADO e ZUFFO, 2003).

O problema da detecção de colisão entre objetos é fundamental em qualquer simulação do mundo físico, sendo estudado por diversas comunidades, incluindo a robótica, a computação gráfica e a geometria computacional. Diante disso, existem diversos métodos e algoritmos para verificar contato entre objetos. Como a execução de algoritmos mais refinados pode causar um custo de processamento maior, soluções mais simples, como a abordagem hierárquica (O'SULLIVAN et al, 2001), normalmente são adotadas para identificar uma colisão: inicialmente algoritmos mais simples são empregados sem fornecer níveis altos de precisão. Uma vez identificada uma colisão com esses métodos, métodos de refinamento são propostos a fim de fornecer a precisão desejada. A seguir são citadas algumas técnicas de colisão encontradas na literatura que utilizam a abordagem hierárquica.

3.1 Abordagem Hierárquica (uso de volumes limites)

A abordagem hierárquica, ou uso de volumes limites é utilizado quando há uma necessidade de uma rápida detecção de colisão. Volume limite é a representação da extensão espacial máxima de um objeto, podendo ser representada por um cubo ou esfera. O uso deste tipo de abordagem consiste em criar aproximações geométricas para objetos complexos, permitindo eliminar áreas dos objetos que não estão em contato (RIQUELME, 2005). O'Sullivan et al (2001) descreve algumas dessas abordagens sendo elas *octrees, AABB-trees, OBB-trees e k-dops.*

OCTREES

Segundo Hearn e Baker (1997) *octrees* são estruturas de árvores hierárquicas usadas para representar objetos sólidos em alguns sistemas gráficos. A estrutura é organizada para que cada nó corresponda a uma região do espaço tridimensional. Esta representação para sólidos tem a vantagem de reduzir o armazenamento de informação para objetos tridimensionais. O esquema de codificação divide regiões de espaços tridimensionais, normalmente cubos, em octantes e guarda elementos em cada nó da árvore. Quando os voxels[2] do octante são do mesmo tipo, este valor é guardado no elemento especifico do nó. Se estes forem vazios, o nó é tido como vazio, se os voxels forem heterogêneos, o octante é subdividido em outro octante, e assim por diante.

A Figura 17 mostra um exemplo da abordagem de *octrees*, exibindo a divisão do volume limite, representado por uma caixa em oito partes.

[2] **voxel** (combinação entre as palavras *volumetric* e *pixel*) é um elemento de volume, representando um valor do espaço tridimensional. Palavra análoga ao pixel representando os dados de uma imagem 2D.

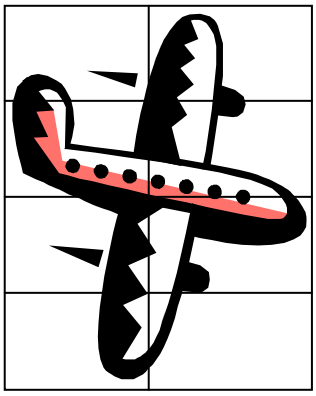


Figura 17: Exemplo da Abordagem Hierárquica Octree (RIQUELME, 2005).

Axis-aligned bounding boxes

Hudson et al (1997) e Klosowski et al (1998) apresentaram o algoritmo *axis-aligned bounding boxes (AABB)*, que checa os dois pontos mais afastados do objeto e determina uma "caixa" ao redor do mesmo, conforme exemplifica a Figura 18(a). O problema desse algoritmo é a fidelidade em relação ao tamanho do objeto, não provendo uma detecção de colisão muito precisa.

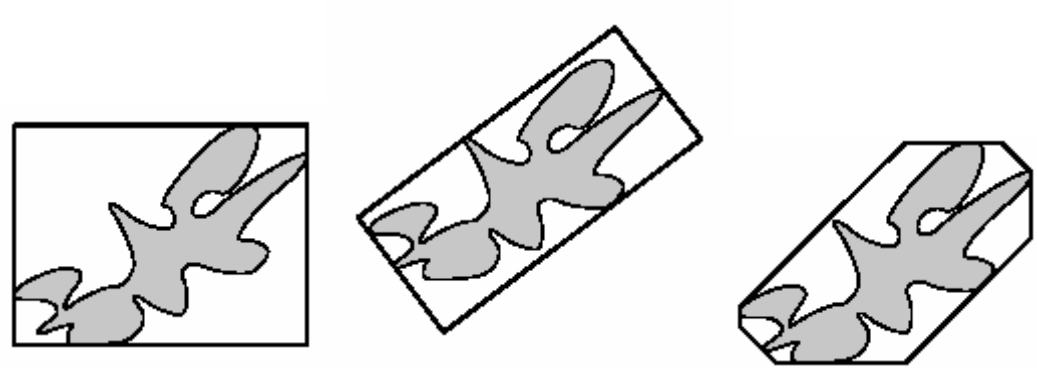
Oriented Bounding Box

Uma melhoria do algoritmo anterior é apresentada pelos pesquisadores, no algoritmo denominado O*riented Bounding Box (OBB)*, que detecta todos os pontos mais afastados do objeto para definir uma borda, fazendo com que essa "caixa" fique com uma área muito mais próxima do objeto, conforme ilustra a Figura 18(b). Esse algoritmo é um pouco mais eficiente que o AABB porque sua área de colisão é menor que a proporcionada por aquele algoritmo. Mesmo assim, ainda não propicia uma fidedignidade ideal ao formato do objeto.

K-dops

Klosowski et al (1999) propuseram ainda o algoritmo *k-dops*, que considera um número específico de faces pré-determinadas, designado por uma varíavel **k**. O algoritmo busca os

pontos mais “afastados” do objeto e coloca as faces pré-determinadas perpendiculares a esses pontos fazendo com que a sua área de colisão diminua. Para esse tipo de polígono foi dado o nome de *k-dops* (*discrete orientation polytopes*). A Figura 18(c) representa a borda determinada por esta técnica com k = 8.



(a) (b) (c)

Figura 18 – Representação de bordas para detecção de colisão: (a) algoritmo AABB; (b) algoritmo OBB; (c) algoritmo K-Dops.

3.2 Detecção de colisão em objetos rígidos e deformáveis

Segundo Lau et al (2002), a detecção de colisão está dividida em dois grandes grupos: detecção de colisão em objetos rígidos e detecção de colisão em objetos deformáveis ou de superfície flexivel. A detecção de colisão em objetos rígidos envolve o teste de penetração de um objeto em outro. Já a detecção de colisão em objetos deformáveis trata, além da interação entre os objetos, das deformações causadas por esta interação.

Entre as soluções para objetos rígidos, Lau et al. (2002) citam a subdivisão hierárquica do espaço, a subdivisão hierárquica do objeto e o cálculo incremental de distância.

Na subdivisão hierárquica do espaço, os objetos são juntados hierarquicamente de acordo com a região que se encontram no ambiente. Quando o objeto muda de posição, apenas objetos ou primitivas que estão na subdivisão do ambiente para o qual o objeto está se

locomovendo são checados para a detecção de colisão.

Na subdivisão hierárquica do objeto, cada elemento é subdividido hierarquicamente em uma área e a colisão é checada quando há uma intersecção dessas áreas pré-determinadas.

No cálculo incremental de distância, a distância mínima entre dois objetos é gradualmente "rastreada" e a detecção de colisão pode ser determinada eficientemente.

Lau et al (2002) afirmam também que os métodos de colisão em objetos deformáveis, por sua vez, são baseados na subdivisão hierárquica do objeto que pode ser de dois tipos: um destinado para objetos representados por polígonos encadeados e outro por superfícies paramétricas (mapeamentos de algum subconjunto do plano ao espaço).

Objetos representados por polígonos encadeados são usualmente deformados, alterando as posições dos vértices dos polígonos. Já os representados por superfícies paramétricas são, geralmente, deformados pela alteração do controle de pontos da superfície.

3.3 Métodos de detecção de colisão

He e Kaufman (1997) também apresentaram uma proposta para a detecção de colisão para objetos volumétricos. No primeiro passo é associado um índice para cada nó com a mínima e máxima probabilidade de colisão na superfície de todos os pontos dentro do cubo (Figura 19c). Observa-se que os nós de mesmo nível podem se sobrepor por estar sendo utilizada uma *sphere tree*. O segundo passo é eliminar os nós sólidos e vazios; o usuário fornece um percentual máximo **tmax** de preenchimento que são os nós sólidos, e um percentual mínimo **tmin** que são os nós vazios. Todos os nós que tenham tamanho maior ou igual ao **tmax** são considerados totalmente sólidos; aqueles com tamanho igual ou menor ao **tmin** são considerados totalmente vazios (Figura 19b e Figura 19a). O espaço definido para detecção de colisão tem que ser refeito dinamicamente, pois podem ocorrer transformações no objeto devido à detecção de colisão já

realizada, havendo modificações na sua área. Normalmente aumentando o número de nós vazios e diminuindo o número de nós sólidos a complexidade da árvore diminui, aumentando a velocidade da detecção de colisão. O algoritmo começa do nível raiz (Figura 19d) prosseguindo até o nível que todos os nós são considerados sólidos.

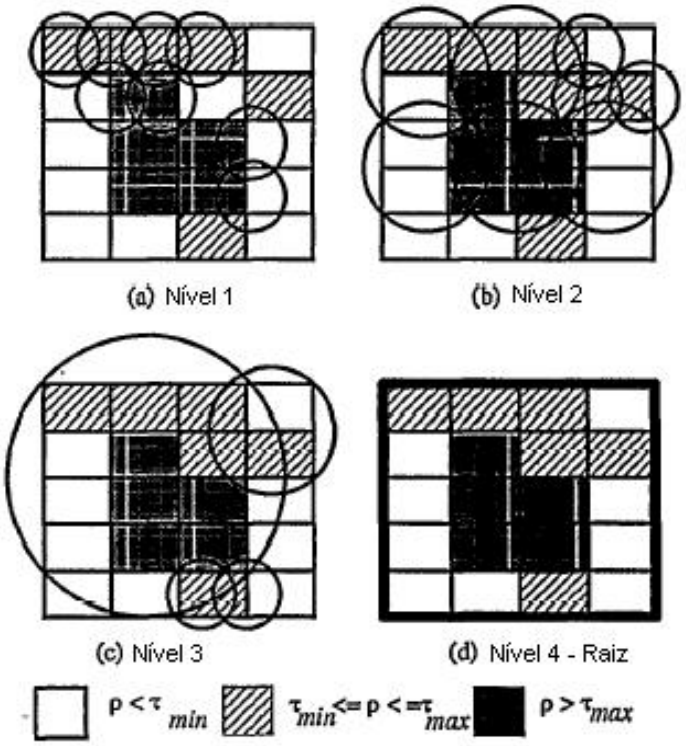


Figura 19 - Representação da construção da árvore *spheretrees*. (He e Kaufman,1997)

Outra solução apresentada por He (1999) é a detecção de colisão usando árvores QuOSPO (*Quantized Orientation Slabs with Primary Orientations*), que faz a mensuração de formas indefinidas a partir de orientações primárias. O algoritmo aproxima a área de detecção de colisão de um objeto deformado, utilizando as suas orientações primárias e traçando linhas que tenham contato com o objeto, tornando essa área mais fiel à modelagem do objeto e a detecção de colisão mais rápida e precisa. A Figura 20 ilustra uma representação deste algoritmo.

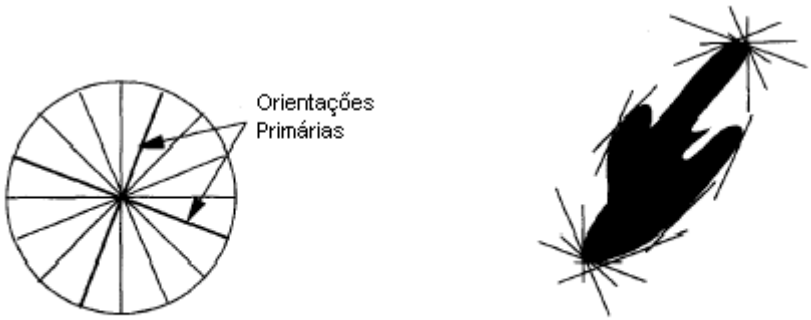


Figura 20 - Representação do Algoritmo QuOSPO. (He e Kaufman,1997)

No projeto V-COLLIDE Hudson et al (1997) fazem uma combinação de recursos oferecidos pelas bibliotecas I-COLLIDE (COHEN et al,1995) e RAPID (GOTTSCHALK et al, 1996) para estender a detecção de colisão presente no VRML 2.0. Num primeiro momento, algoritmos presentes na I-COLLIDE são usados para filtrar colisões entre um grande número de objetos, determinando pares de objetos que estejam potencialmente colidindo.

Em um segundo momento, algoritmos da biblioteca RAPID são aplicados para determinar se os objetos recebidos do primeiro estágio estão realmente colidindo.

O estudo dos principais algoritmos faz com sejam percebidas algumas soluções viáveis para o problema de detecção de colisão em objetos deformáveis não regulares, que é o foco principal deste projeto. Com esses estudos foi notado que a utilização de um método hierárquico poderia trazer uma melhor aproximação da área de colisão ao objeto, tendo, assim, um melhor refinamento e desempenho. Ao escolher a abordagem hierárquica foi desenvolvido um método descrito no capítulo 4 que utiliza técnidas nativos da API J3D juntamente com o conceito de *octrees.*

CAPÍTULO 4 – IMPLEMENTAÇÃO DE COLISÃO UTILIZANDO HIERARQUIAS EM JAVA3D.

A partir dos estudos de algoritmos de detecção de colisão foi escolhida a utilização de uma abordagem hierárquica, que consiste em utilizar métodos iniciais simples para a detecção de colisão e, a partir dessa detecção, aplicar se um novo método para que o resultado seja refinado. Para o desenvolvimento desse projeto foi necessária a pesquisa de métodos que a API J3D fornece, sendo estudados os métodos nativos *BoundingBox* (BB), *BoundingSphere* (BS) e Detecção de colisão por faces (DCF). O próximo passo foi o estudo do conceito de *octrees*, escolhido para o desenvolvimento desse projeto.

Para a fase inicial do projeto, foi desenvolvido um AV simples em J3D, através da classe *SimpleUniverse*. Foram inseridas duas primitivas do J3D (uma esfera e um cubo), uma com movimentação simples através do mouse e outra estática, no qual foi adicionada a detecção de colisão (Figura 21). Nesse primeiro momento a área de detecção de colisão era responsável para demonstrar a colisão entre as primitivas com métodos já implementados na API J3D: *BoundingBox*, *BoundingSphere* e Colisão por Faces (Figura 22). Os códigos das implementações são apresentados nos apêndices A, B, C, D e E.

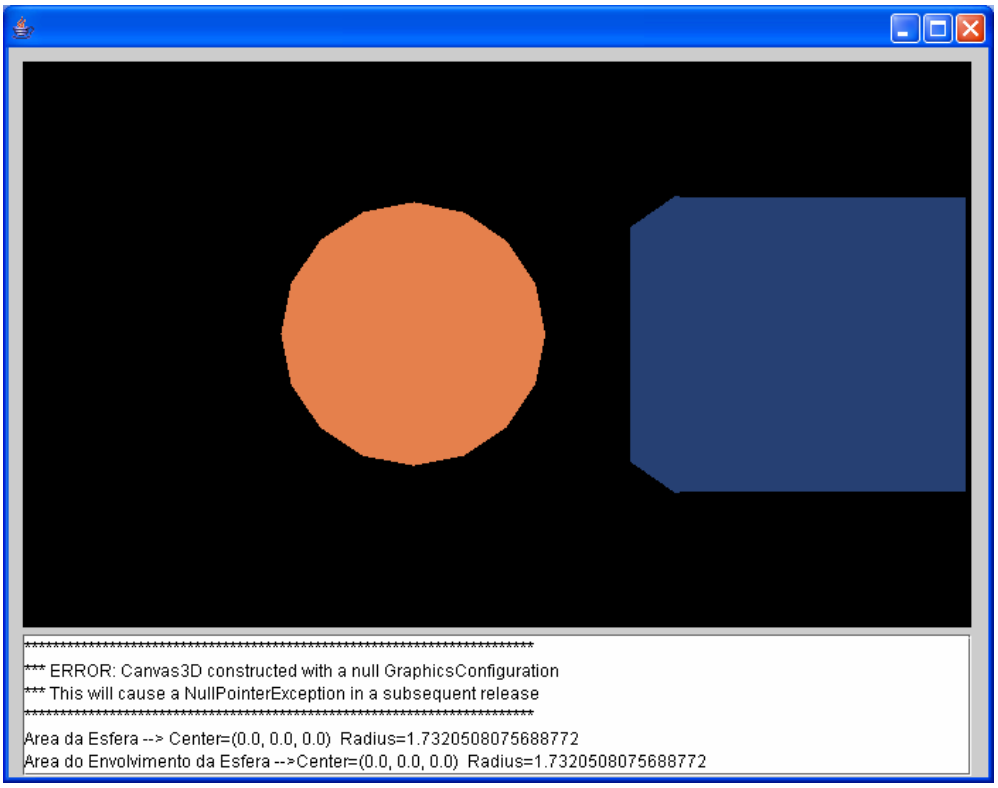


Figura 21 – Demonstração do ambiente com primitivas simples.

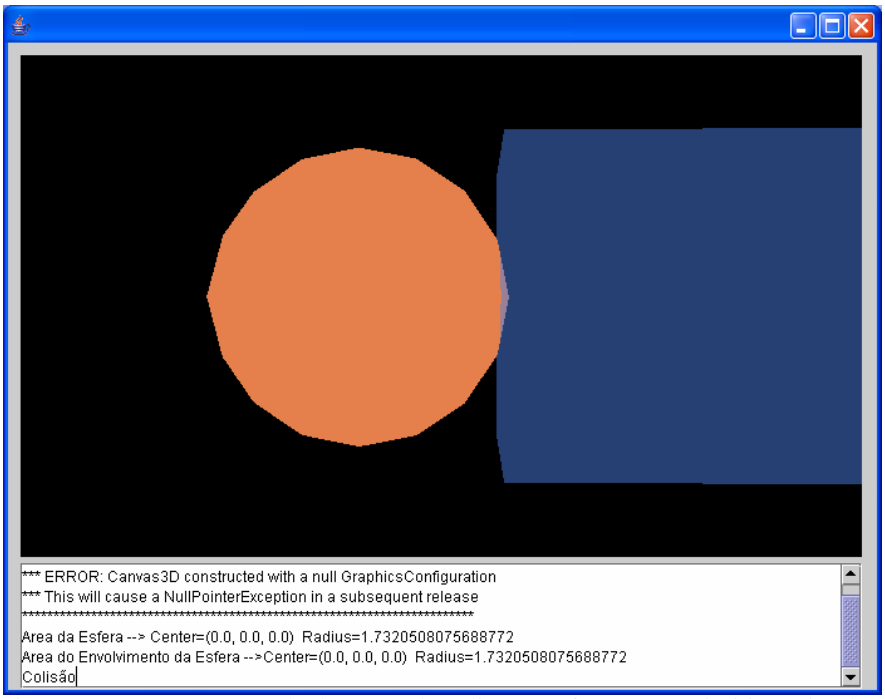


Figura 22 – Demontração da detecção de colisão por faces com primitivas simples.

Na segunda fase do projeto foi desenvolvido um AV um pouco mais complexo, no qual foram importados dois arquivos previamente modelados no 3DStudioMax (AUTODESK, 2005) que consistem em modelos utilizados em um protótipo de aplicação para o treinamento médico do exame de punção de mama (LIMA et al, 2004). Da mesma forma, em um desses objetos (seringa) foi adicionado métodos de movimentação por mouse, e o outro objeto (mama) ficou estático na cena. Ao objeto estático foi adicionada a detecção de colisão. Da mesma maneira, nesta fase foram implementados apenas os métodos nativos da API J3D (Figura 23). As implementações realizadas podem ser observadas nos apêndices E e F.

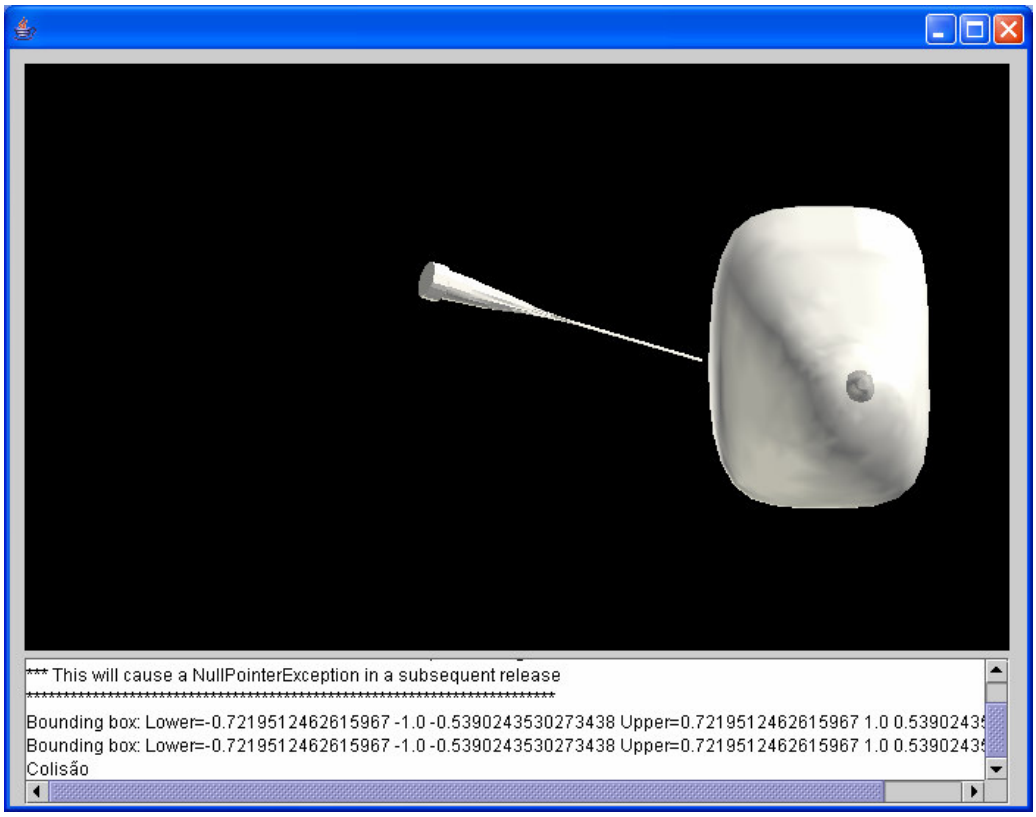


Figura 23 – Demonstração de colisão com método *BoundingBox*.

Com os estudos desses métodos foi detectado que o método BB (Figura 24), que obtém os dois pontos mais distantes do objeto, envolvendo-o com uma caixa (Figura 25), a área que

envolve o objeto era muito distante do objeto em si, não provendo uma precisão adequada para uma ferramenta de aplicação médica (Figura 26), mas mesmo com essa falta de precisão, o método *BoundingBox* possui a vantagem de ter um desempenho aplicável a uma ferramenta de RV para treinamento médico.

```
                CollisionDetector collisionDetector = new CollisionDetector(box);
                collisionDetector.setSchedulingBounds(box.getBounds());
                objRoot.addChild(collisionDetector);

wCriterion[0] = new WakeupOnCollisionEntry(shape, WakeupOnCollisionEntry.USE_BOUNDS);
wCriterion[1] = new WakeupOnCollisionExit(shape,WakeupOnCollisionExit.USE_BOUNDS);
wCriterion[2] = new WakeupOnCollisionMovement(shape,WakeupOnCollisionMovement.USE_BOUNDS);
```

Figura 24 – Trecho de código do *BoundingBox*

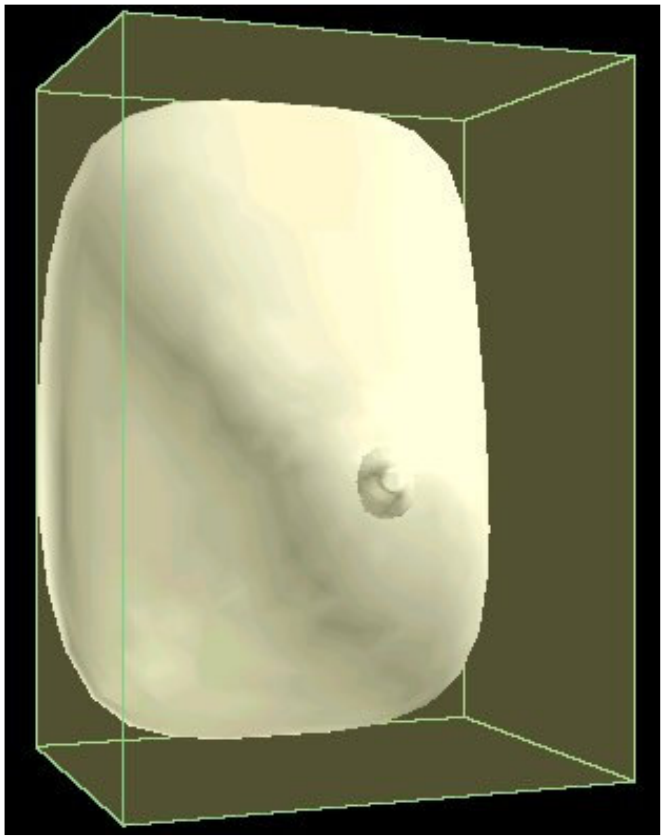


Figura 25 – Demonstração do método *BoundingBox*

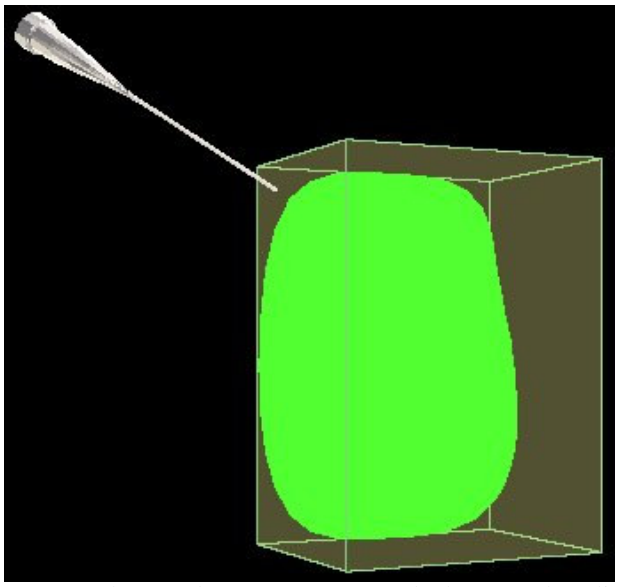


Figura 26 – Demonstração do método BB no momento da detecção de colisão com a seringa.

O segundo método estudado foi o BS, que tem as mesmas características que o método BB, mas ao invés de envolver o objeto com um cubo, este é envolvido com uma esfera (Figura 27). O ponto mais afastado do objeto é detectado e a distância do centro do objeto até esse ponto é o raio da esfera desenhada. A Figura 28 ilustra como essa esfera envolve o objeto. Do mesmo modo que o método BB, nesse método há uma defasagem no quesito de precisão, pois em objetos com formas irregulares partes desse objeto podem estar distantes da área de colisão determinada pelo método. A Figura 29 demonstra essa defasagem de precisão.

```
//coloca obj na colisão
CollisionDetector1 cd = new CollisionDetector1(shapeObj);
BoundingSphere bounds = new BoundingSphere();
bounds.set(shapeObj.getBounds());
shapeObj.setCollisionBounds(bounds);
cd.setSchedulingBounds(bounds);
objTrans.addChild(cd);
```

Figura 27 – Trecho de código do *BoundingSphere*

Figura 28 – Demonstração do método *BoundingSphere*

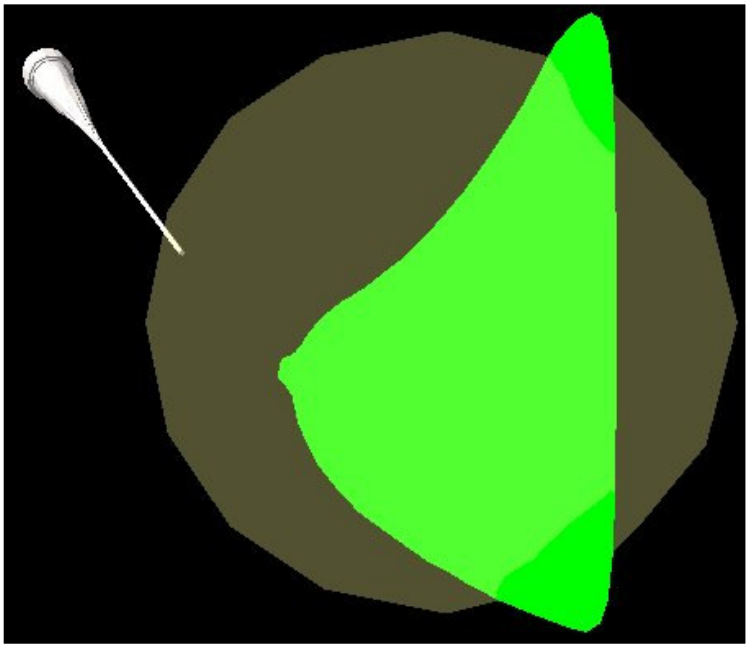


Figura 29 – Demonstração de Colisão com o método *BoundingSphere*

O terceiro método estudado implementado pelo J3D é o método de detecção de colisão por faces (DCF). A Figura 30 mostra o trecho de código do método DCF. Esse método é muito interessante, pois nele pode se observar um comportamento no J3D não verificado

anteriormente. Com a J3D a detecção de colisão é feita hierarquicamente, ou seja, quando se indica na classe que o método de detecção de colisão a ser utilizado é o DCF é solicitado que se instancie uma área de colisão, podendo ser utilizada o BB ou o BS. A partir disso, apenas é considerado a DCF após a quebra dos limites do método mais simples, previamente estabelecido.

```
wEnter = new WakeupOnCollisionEntry(shape,WakeupOnCollisionEntry.USE_GEOMETRY);
wExit = new WakeupOnCollisionExit(shape,WakeupOnCollisionExit.USE_GEOMETRY);
wakeupOn(wEnter);
```

Figura 30 – Trecho de Código do médoto de Detecção de Colisão por Faces.

A partir da verificação do comportamento dos métodos nativos da J3D foi iniciada a fase de implementação de um método que pudesse manter o desempenho dos métodos *BoundingBox e BoundingSphere* da API Java3D e melhorar a precisão dos mesmos. A técnica implementada utiliza o conceito de hierarquia iniciando a detecção por métodos fornecidos pela API Java3D e, quando as barreiras desses primeiros métodos são vencidas, métodos mais avançados são aplicados. A Figura 31 mostra o fluxograma da implementação realizada.

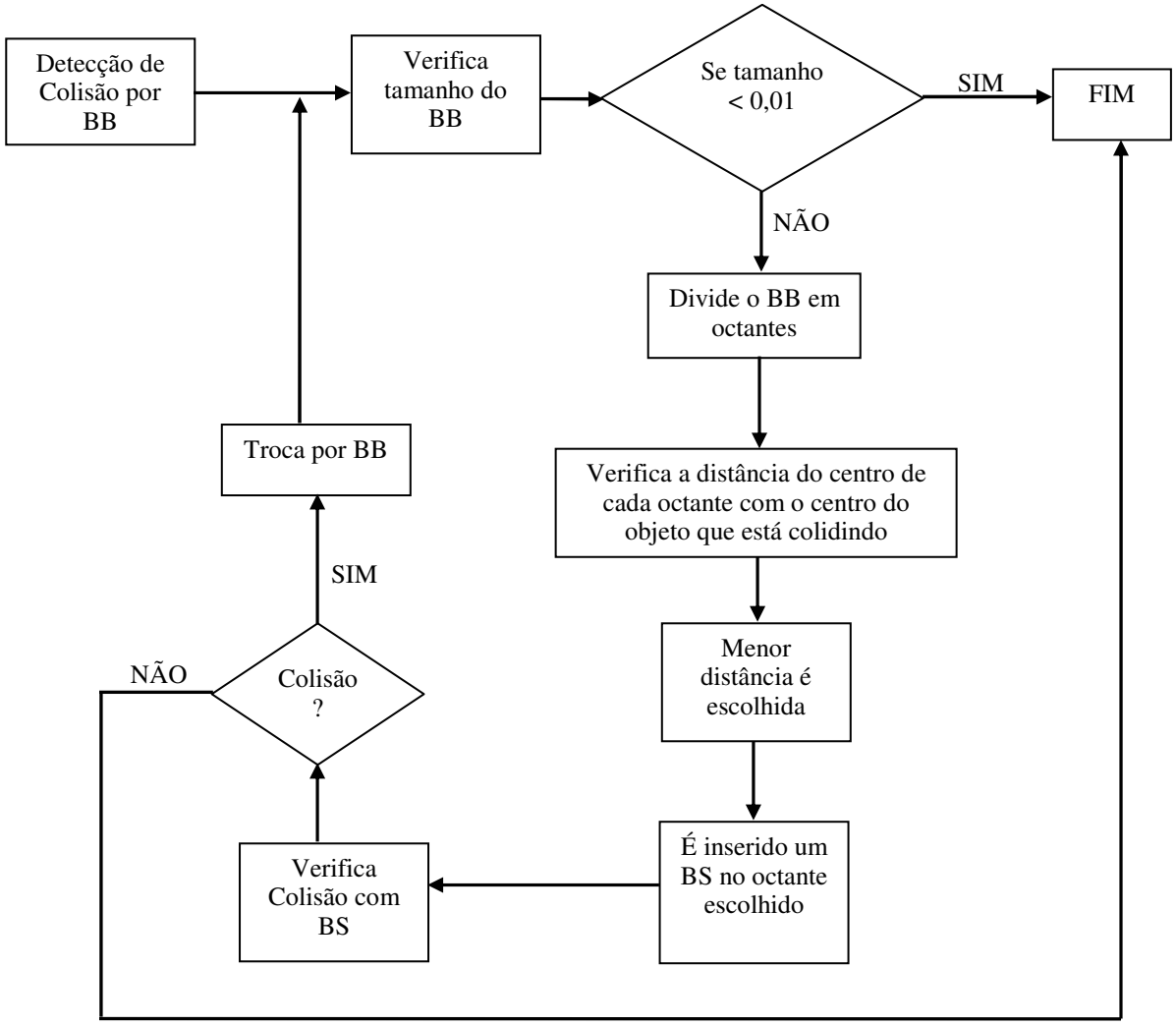


Figura 31 – Fluxograma da implementação do método

O método desenvolvido utiliza o objeto que foi definido como estático, tendo a técnica BB como base. O método simplesmente detecta os pontos afastados do objeto (mama) e envolve esse objeto com uma caixa, sendo essa caixa o primeiro limite de colisão do objeto. Quando o método detecta colisão pelo BB, o mesmo obtém os pontos mais afastados (*upper* e *lower*) desse BB e define as outras caixas que irão formar o *octree*. O *octree* é definido por um vetor de oito posições que guarda os pontos necessários para a formação de um novo BB.

Após a divisão do BB em oito partes é verificada a distância entre o centro de cada octante com o centro do objeto que está colidindo. O octante cujo centro apresenta a menor distância euclidiana (equação 1) em relação ao centro do objeto que está colidindo é escolhido para os refinamentos sucessivos. Na equação 1, CA e CM são Coordenadas centrais da agulha (CA) e Coordenadas centrais da Mama (CM). Este octante é envolvido com um *BoundingSphere* e com esse método é verificada novamente a colisão. Se a colisão não for identificada, o método é finalizado, passando esta informação à aplicação. Caso contrário, o mesmo processo é executado recursivamente, até que o octante chegue a um tamanho mínimo pré-determinado (colisão detectada) ou esse octante esteja vazio (colisão não detectada). O tamanho mínimo do octante é calculado pela distância euclidiana entre os extremos desse octante (Figura 32 e Figura 33). O código pode ser visto nos apêndices G, H, I e J.

$$d(CA,CM) = \sqrt{(CA_x - CM_x)^2 + (CA_y - CM_y)^2 + (CA_z - CM_z)^2} \quad (1)$$

onde:

d = distância entre coordenadas centrais da agulha e da mama
CA = Coordenadas centrais da agulha
CM = Coordenadas centrais da mama

```
    Colisão_Detectada
    Adiquire pontos afastados do BB (upper, lower)
    Se tamanho do BB < 0,01 então
            FIM;
    Senão
            Aloca um vetor de oito posições
            Divide o BB em oito parte
            Coloca o valor dos pontos afastados de cada octante no vetor
              Verifica distância do centro dos octantes com centro do objeto que colidiu
            Escolhe o octante com a menor distância
            Envolve octante com BS para diminnuir a área de colisão
            Se não houver colisão
                    FIM
            Senão
                Envolve octante com BB novamente para nova divisão
            Fim senão
    Fim senão
FIM
```

Figura 32 – Algoritmo do método desenvolvido

A Figura 33 mostra bidimensionalmente o funcionamento do método, para facilitar a compreensão da técnica. Salienta-se que no espaço bidimensional, como é possível perceber na Figura, são representados quadrantes. No espaço 3D esses quadrantes se referem a octantes, havendo mais quatro dimensões além das representadas.

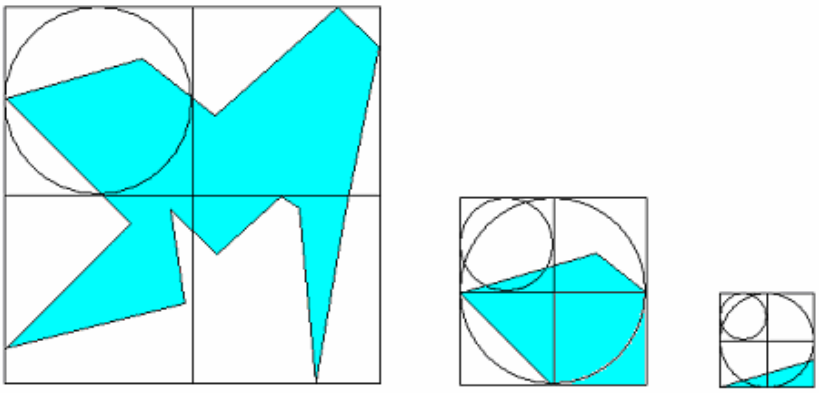


Figura 33 – ilustração do método em desenvolvimento em 2D.

Foi possível notar que durante o desenvolvimento que a implementação do algoritmo não é muito complexa, mas a ecolha do AV é de suma importância para que a aplicação funcione coretamente. Os resultados obtidos após a implementação são apresentados no capítulo 5.

CAPÍTULO 5 – RESULTADOS

Através dos testes realizados com um PC com processador Pentium 4 3.0 HT, placa de vídeo Gforce FX 5200 e 512 de RAM e uma classe que captura a quandidade de quadros exibidos na tela por segundo (Apêndice J), foi virificado que o método *BoundingBox* fornece alto desempenho, mas não oferece fidelidade em relação à forma do objeto, pois é uma "caixa" que envolve o objeto. Os limites dessa "caixa" podem estar muito distantes do objeto, quando este possui forma irregular, não provendo uma detecção de colisão muito precisa, que é necessário para aplicações de treinamento médico.

O gráfico da Figura 34 mostra o desempenho do método *BoundingBox.* Verifica-se que depois de 3 segundos a aplicação se estabiliza e o método apresenta uma taxa de quadros relativamente alta, entre 59 e 60 FPS (frames por segundo). Quando a taxa de FPS é menor que 59,5 houve uma detecção de colisão no AV e quando esta taxa é superior a 59,5 são condições normais da aplicação, ou seja, o objeto está interagindo com o ambiente sem nenhuma intervenção.

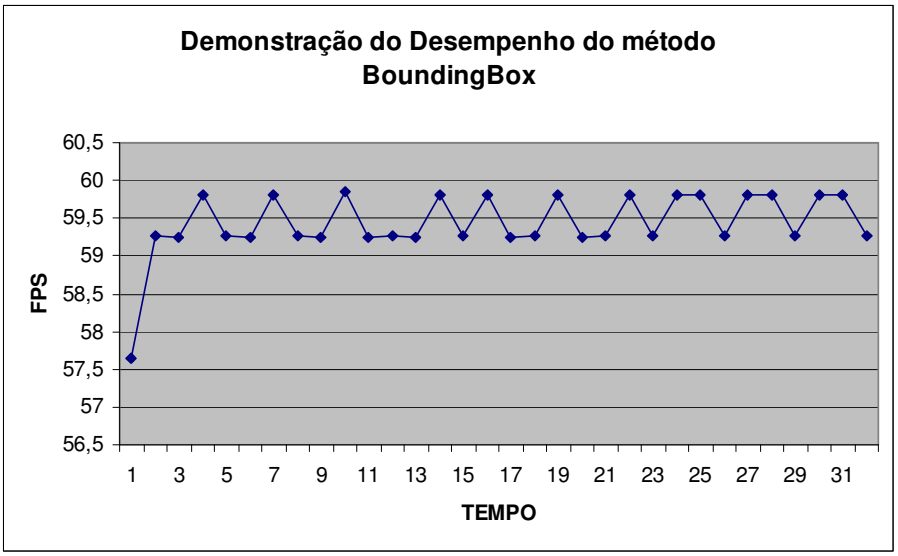


Figura 34. – Demonstração do Desempenho do Método *BoundingBox.*

No método *BoundingSphere* há também a vantagem do desempenho (Figura 35), pois as respostas ocorrem em tempo adequado como o fornecido pelo método anterior. O gráfico também mostra o desempenho do método *BoundingSphere* relacionando a quantidade de quadros exibidos em um segundo (FPS). Da mesma forma que ocorreu no caso anterior, o momento da colisão indica uma taxa de 59,5 FPS.

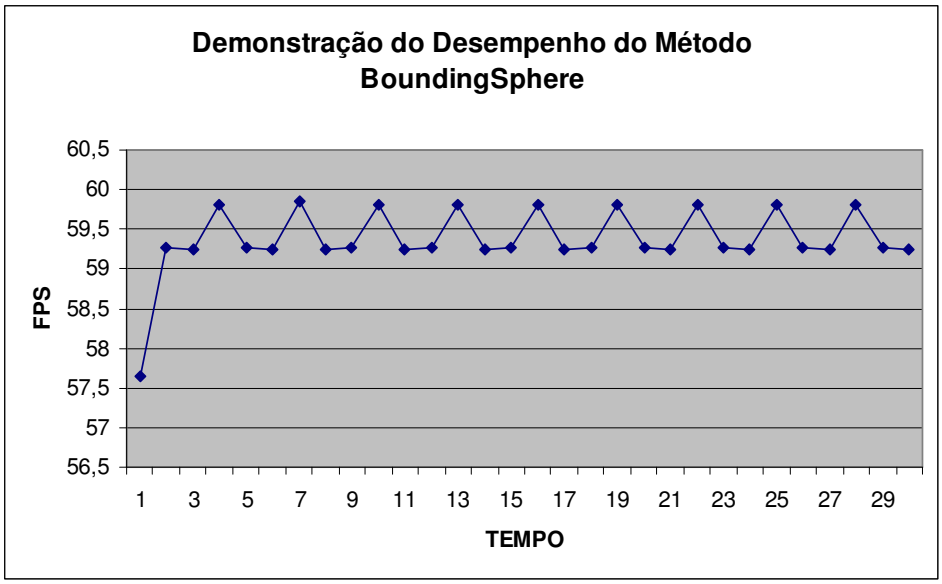


Figura 35 – Demonstração do Desempenho do método BoundingSphere

No entanto, ainda há falta de precisão devido à forma que a área de detecção de colisão possui, visto que apenas o desempenho não é suficiente para uma aplicação médica. No método *BoundingSphere* o objeto é envolvido por uma esfera determinada em relação aos pontos mais afastados do mesmo. Assim, em objetos com formatos irregulares muitos pontos podem ficar longe das fronteiras da esfera utilizada (Figura 36).

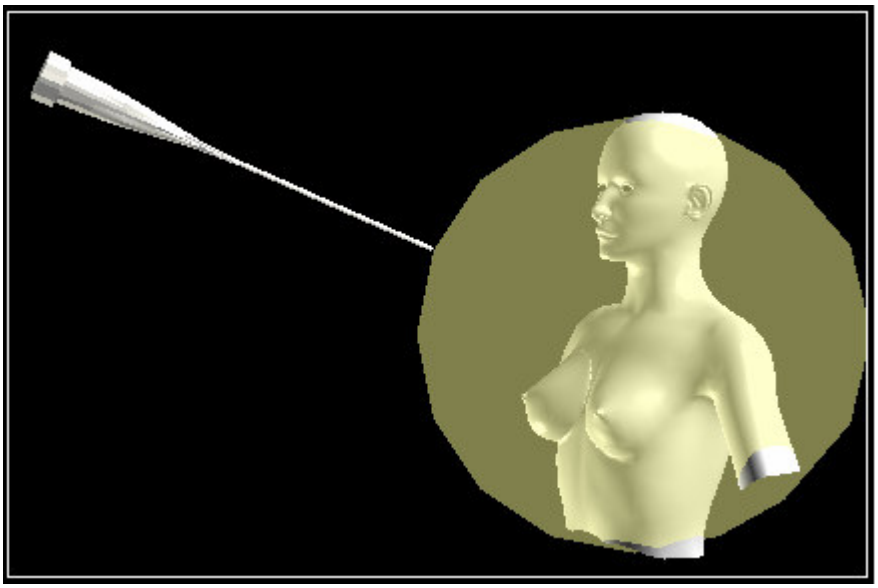


Figura 36 – Colisão com BS em um objeto irregular.

O gráfico da Figura 37 demonstra o desempenho do método DCF, podendo ser notado que durante a execução da aplicação, antes do objeto entrar nos limites de colisão, a aplicação tem uma taxa de desempenho satisfatória (entre 59 e 60 FPS), mas quando é detectada a colisão essa taxa cai para quase zero, fazendo com que a aplicação fique lenta e praticamente estacionada. Nota-se que a queda da taxa mostrada no gráfico é dada quando ocorre uma detecção de colisão. O gráfico monstra, ainda, que para a aplicação se reestabelecer há um tempo em torno de 4 a 5 segundos.

No entanto, observou-se que a aplicação fica paralisada durante aproximadamente um minuto e só após esse tempo é que os FPS começam a ser contabilizado. Utilizando por padrão a configuração de hierarquia da Java3D, o método DCF proporciona uma solução perfeita em relação à precisão, pois a colisão só é detectada quando o outro objeto que colidiu atinge uma das faces do objeto que espera a colisão. Observa-se, entretanto, que o desempenho é extremamente baixo, pois após a identificação do espaço pré-determinado, a aplicação torna-se tão lenta que é praticamente impossível interagir com a mesma.

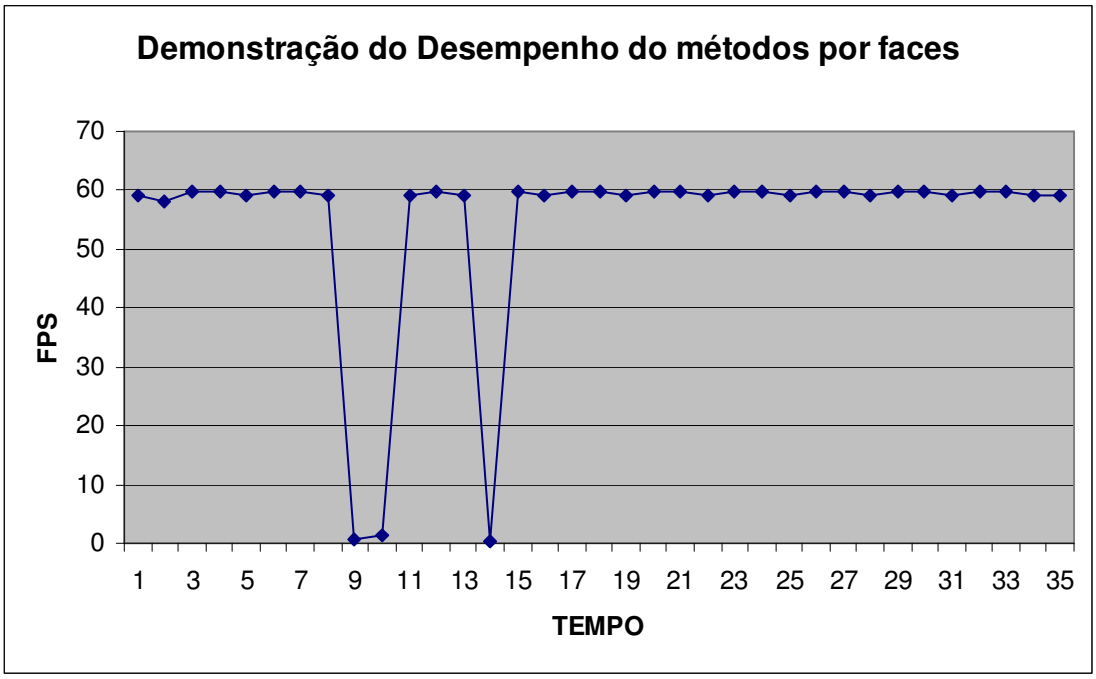


Figura 37 – Demonstração do desempenho da Detecção de Colisão por Faces

O método desenvolvido utiliza as vantagens de desempenho dos métodos nativos da API Java e uma estratégia que permite a detecção de colisão com precisão, sem diminuição significativa de desempenho. Com os testes realizados verificou–se que o algoritmo desenvolvido não apresentou perda de desempenho notória em relação aos métodos nativos da API J3D. O gráfico da Figura 38 demonstra o desempenho do método desenvolvido que detecta colisão usando hierarquias através da técnica implementada neste trabalho, mostrando a quantidade de quadros em um segundo são apresentados na tela.

Do mesmo modo que foi observado nos métodos anteriores, o gráfico demonstra que, depois de atingida a estabilidade do aplicativo, a quantidade de frames que está abaixo de 59,5 é referente ao momento em que foi detectada colisão no ambiente.

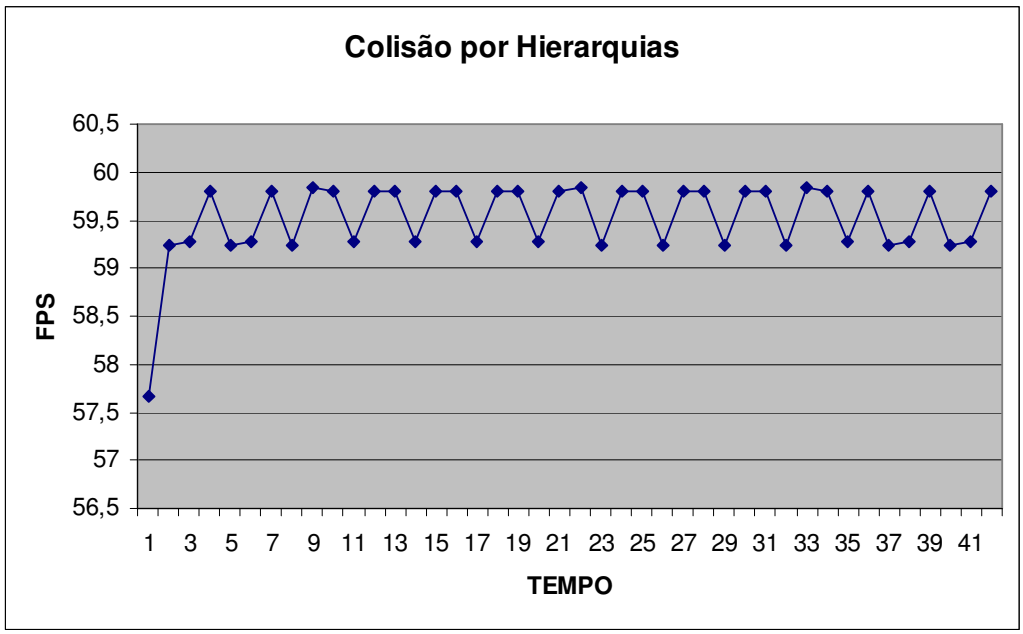


Figura 38 – Demonstração do desempenho do método desenvolvido.

Para demonstrar a eficiência no desempenho do método desenvolvido, que é praticamente a mesma dos métodos nativos a da API J3D, foi realizada uma comparação entre os métodos citados, apresentada no gráfico da Figura 39.

Foi observado que os métodos *BoundingBox* e *BoundingSphere* têm praticamente o mesmo desempenho e o método desenvolvido tem uma leve perda de desempenho, mas nada tão significante que a aplicação não possa ser utilizada em aplicações de RV para treinamento médico.

Neste gráfico a linha azul demonstra o desempenho do método *BoundingBox*, a linha rosa demonstra o desempenho do método *BoundingSphere*, e a linha amarela demonstra o desempenho do método desenvolvido. Nota-se que, depois de atingida uma estabilidade na aplicação, nenhum dos métodos obteve uma taxa de FPS mais baixa que 59. Apenas em momentos em que foi detectada a colisão no ambiente é que a taxa de frames por segundo esteve abaixo de 59,5. Nos casos normais em que ocorreu apenas a navegação no ambiente, os métodos

apresentam uma taxa superior a 59,5 FPS e abaixo de 60 FPS, tendo comportamentos similares.

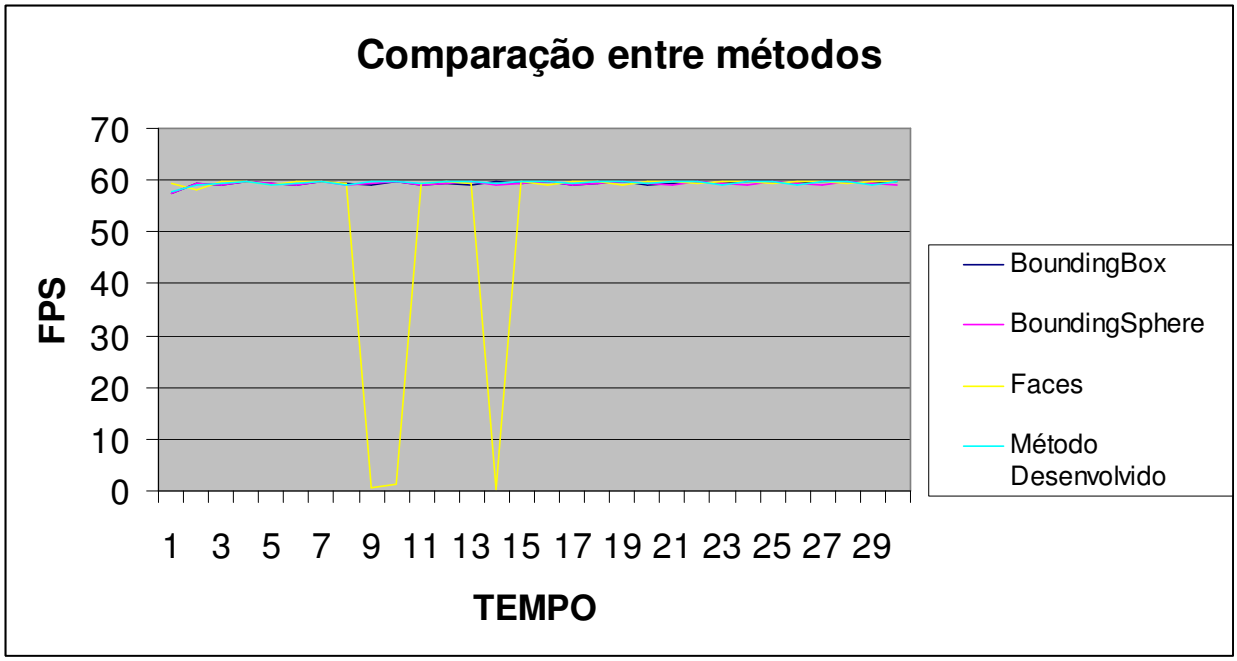


Figura 39 – Comparação de desempenho dos métodos.

Com os testes realizados verificou-se que quando a primeira colisão (com o método BB) é detectada no AV (Figura 40) à distância do objeto que está colidindo é grande em relação ao objeto estático (mama). Com o refinamento por meio do método proposto (Figura 41) a agulha se aproxima mais da mama do que nos métodos nativos. Quanto à colisão é detectada pelo método BS na quinta iteração, os objetos estão bem mais próximos, como pode ser observando na Figura 42. No final do processo, houve efetivamente um encontro entre os objetos (Figura 43).

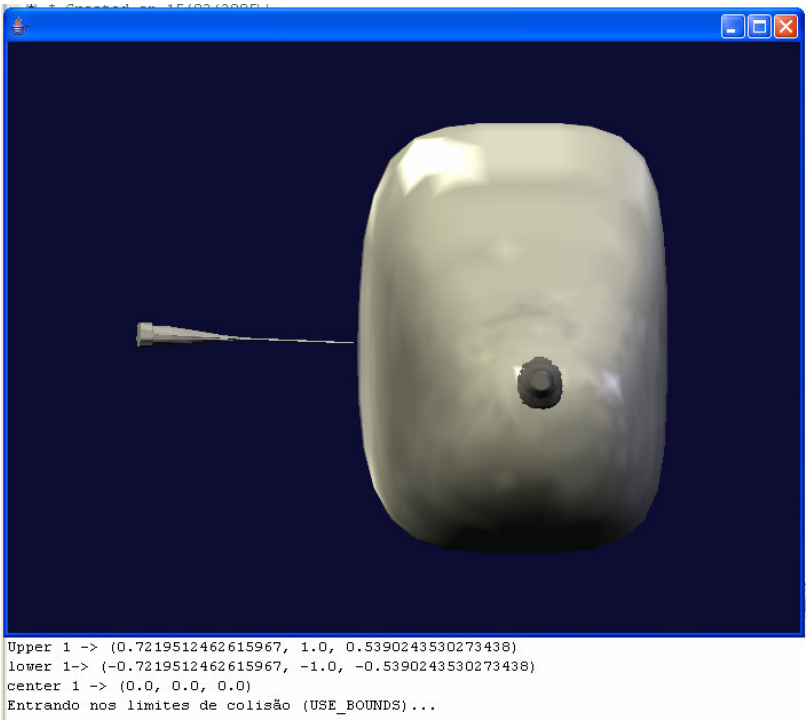


Figura 40 – Demonstração da primeira colisão do método

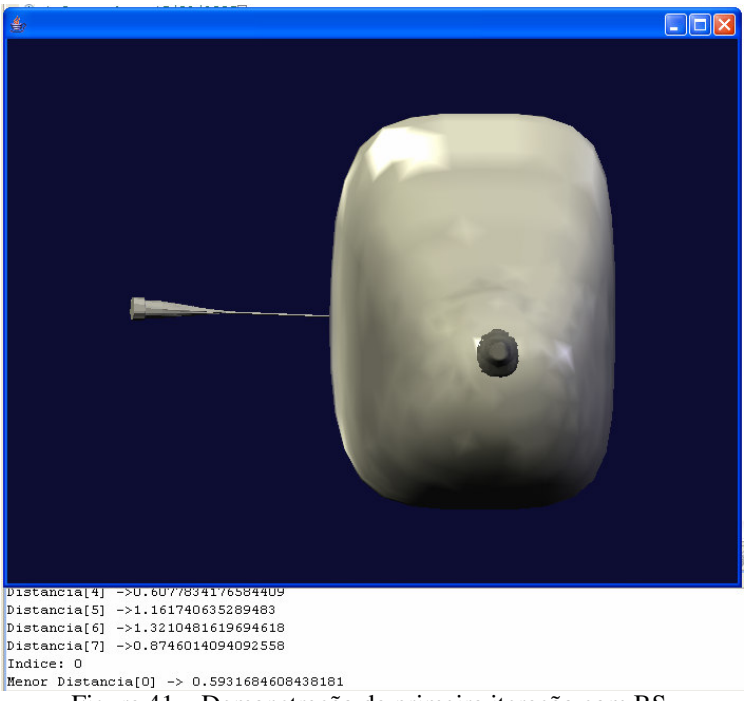


Figura 41 – Demonstração da primeira iteração com BS.

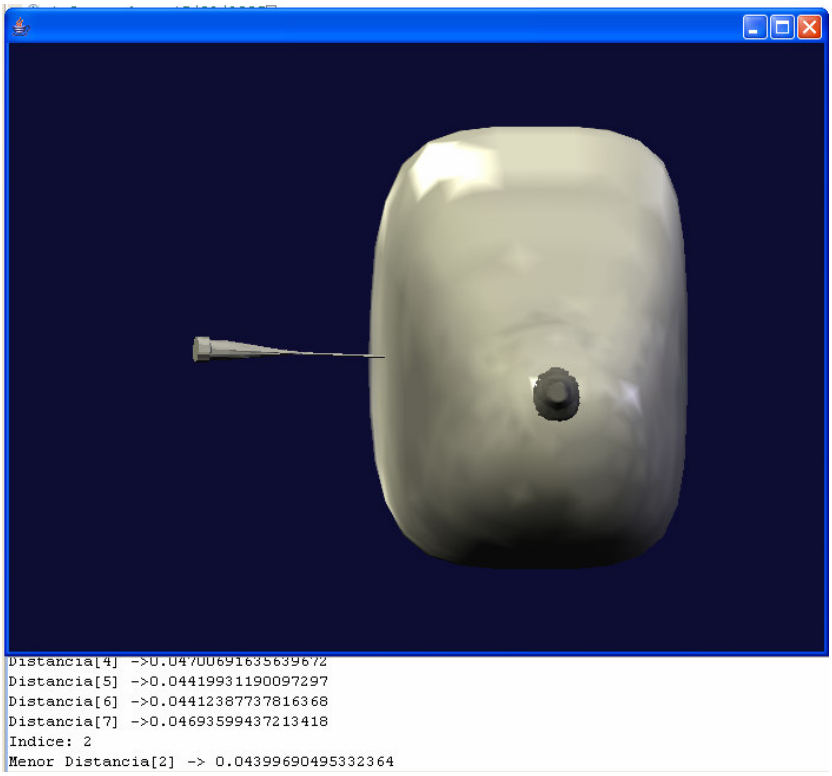


Figura 42 – Distância final da agulha com a mama.

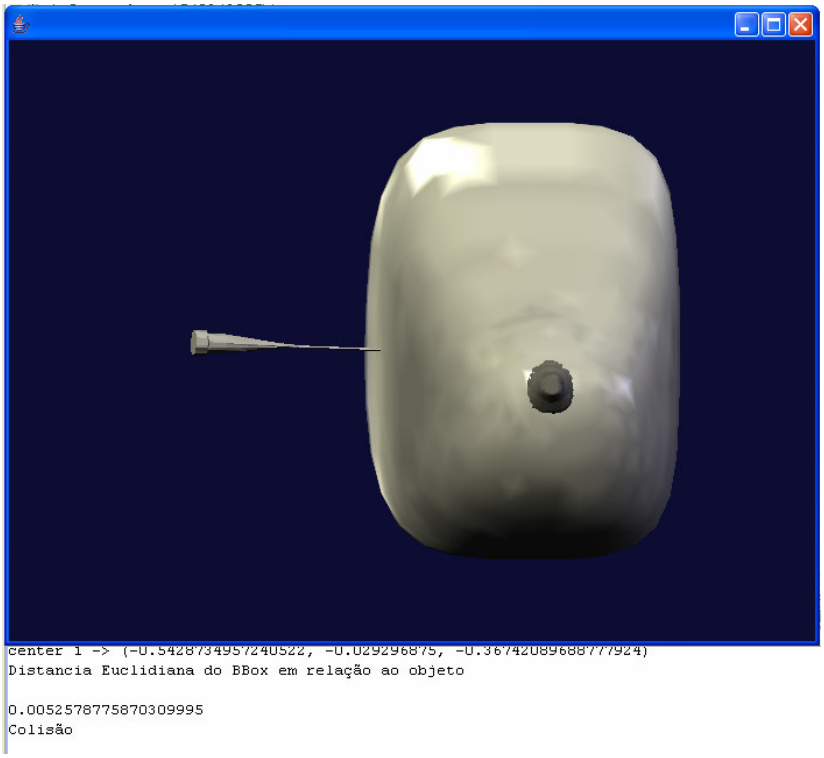


Figura 43 – Indicação de colisão da agulha e da mama

CONCLUSÕES

Este trabalho apresentou algumas abordagens de ferramentas de RV para treinamento médico e uma análise dos métodos oferecidos pela API J3D mostrando suas vantagens e desvantagens. Foi, então, proposto um método de refinamento que não perde o desempenho e oferece um ganho na precisão. Após implementação, foi realizada uma comparação entre os métodos nativos da J3D e o método desenvolvido.

Verificou-se que as classes fornecidas pela API J3D para detecção de colisão não fornecem a precisão necessária para aplicações em ferramentas de treinamento médico, pois a falta de precisão nos métodos *BoundingBox* e *BoundingSphere* são deficiências muito significativas para serem utilizadas em ferramentas de treinamento médico. Por outro lado, apesar da precisão fornecida pelo método DCF, seu custo computacional inviabiliza sua aplicação na área médica, visto que não provê desempenho que possa fornecer resposta em tempo real às ações dos usuários.

O método apresentado, baseado em detecção de colisão utilizando hierarquias, combinando métodos nativos da API J3D acrescido do conceito de *octrees*, proporcionou um ganho na precisão em relação aos métodos da J3D, sem perda significativa de desempenho. Testes mais específicos necessitam ser realizados para a verificar se a precisão alcançada é apropriada para uma ferramenta médica.

Apesar dos estudos realizados serem direcionados para aplicações de treinamento médico, os métodos de refinamento aqui apresentados podem ser usados em outros tipos de aplicações. Assim, este método está sendo transformado em uma classe a fim de que possa ser empregado em ferramentas de RV nas quais a precisão seja um item fundamental, como é o caso de aplicações para treinamento médico.

Como continuidade deste trabalho e com o objetivo de obter um maior grau de precisão

dos refinamentos propostos por meio de uma avaliação mais adequada, alguns pontos podem ser mais aprofundados e novas implementações e testes podem ser realizados, como:

- Testes com outros objetos irregulares, mostrando os momentos de detecção de colisão.
- Rotinas de deformação e de mudança de ponto de vista para aumento do realismo dos AVs gerados.
- Incorporação de dispositivos não convencionais, como luvas e dispositivo háptico, a fim de atingir um maior grau de imersão e interação por parte do usuário.

REFERÊNCIAS BIBLIOGRÁFICAS

AUTODESK. Autodesk 3ds Max. Disponível em: <http://usa.autodesk.com/adsk/servlet/linkedsumindex?siteID=123112&id=5728245&linkID=5604642> acesso em 2005.

AUKSTAKALNIS,S.; BLATNER,D. Silicon Mirage: The Art and Science of Virtual Reality, Peatchpit Press, Berkeley, CA, 1992.

BICHO, A. L.; SILVEIRA JR, L. G.; CRUZ, A. J. A.; RAPOSO, A. B. Programação Gráfica 3D com OpenGL, Open Inventor e Java 3D. **Revista Eletrônica de Iniciação Científica (REIC)**, Brasil, v. 2, n. 1, mar. 2002.

BURDEA, G.; PATOUNAKIS, G.; POPESCU, V.; WEISS, R. E. Virtual Reality-Based Training for Diagnosis of Prostate Cancer. **IEEE Transactions on Biomedical Engineering**, v. 46, n. 10, p. 1253-1260, 1999.

COHEN, J.; LIN, M.; MANOCHA, D.; PONAMGI, M. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments. In: ACM INTERACTIVE 3D GRAPHICS CONFERENCE, 1995. **Proceedings…** 1995, p. 189-196.

GRADECKI, J. The virtual reality construction kit, John Wiley & Sons, 340 Pp., 1995.

GORMAN, P.; KRUMMEL, T.; WEBSTER, R.; SMITH, M; HUTCHENS, D. A Prototype Haptic Lumbar Puncture Simulator. **Medicine Meets Virtual Reality 2000**, IOS Press, 2000.

GOTTSCHALK, S.; LIN, M.; MANOCHA, D. OBB-Tree: A Hierarchical Structure for Rapid Interference Detection. In: ACM SIGGRAPH, 1996. **Proceedings…** 1996, p. 171-180.

HALUCK, R.; WEBSTER, R.; SNYDER, A.; MELKONIAN, M.; MOHLER, B.; DISE, M.; LEFEVER, A. A Virtual Reality Surgical Trainer for Navigation in Laparoscopic Surgery. Medicine Meets Virtual Reality. Studies in Health Technology and Informatics, n. 81, p. 171-176. IOS Press, 2001.

HE, TAOSONG. Fast Collision Detection Using QuOSPO Trees. In: ACM Sysposium on Interactive 3D Graphics, Atlanta, GA USA 1999.

HE, T. KAUFMAN, A. Collision Detection for Volumetric Objects. In: IEEE. National Science Foundation, 1997.

HEARN, DONALD; BAKER M. PAULINE, Computer graphics, C version 2nd ed. 1997 Published by Pretice Hall,Inc.

HUDSON, T. C., LIN, M. C., COHEN, J. GOTTSCHALK, S., MANOCHA, D. V-COLLIDE:

Accelerated Collision Detection for VRML. In: ACM VRML 97, Monterey, CA USA, 1997.

HUNTER, I.; JONES, L.; SAGAR, M.; LAFONTAINE, S.; HUNTER, P. Ophthalmic Microsurgical Robot and Associated Virtual Environment. Computer in Biology and Medicine, v. 25, n. 2, pp-173-182,1995.

KIRNER, C. **Apostila do Ciclo de Palestras de Realidade Virtual**. Atividade do Projeto AVVIC – CNPq (Protem – CC – fase III) – DC/UFSCAR. São Carlos: 1996, p. 1-10.

KLOSOWSKI. J. T., HELD. M., MITCHELL. J.S.B., SOWIZRAL, H., ZIKAN, K. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs. IEEE Transactions on Visualization and Computer Graphics. v. 24, pp. 21-36. January, 1999.

KÜHNAPFEL U., ÇAKMAK H.K., MAAB A. Endoscopic Surgery . IEEE Symposium on Simulation, Delft University, Delft, NL, 13.10.1999 p.22-32.

LAPSIM. University of Michigan Health System: Clinical Simulation Center. Disponível em <http://www.med.umich.edu/umcsc/services/lapsim.htm> acesso em 2005.

LATTA J. N.; OBERG D. J. A Conceptual Virtual Reality Model. **IEEE Computer Graphics and Applications**, v. 14, n. 1, p. 23-28, 1994.

LAU, R.W.H., CHAN, O., LUK, M., LI, F. W. B. A Collision Detection Framework for Deformable Objects. In: ACM VRST'02, p.113-120. November, 2002.

LIMA, L.; NUNES, F. L. S.; BREGA, J. R. F.; SEMENTILLE, A. C.; RODELLO, I. A; TAKASHI, R. Um Protótipo para Simulação de Exame de Punção da Mama Utilizando Realidade Virtual Não Imersiva. In: VII SYMPOSIUM ON VIRTUAL REALITY, 2004, São Paulo. **Anais...** São Paulo, 2004.

MACHADO, L. S.; ZUFFO, M. K.; MORAES, R. M.; LOPES, R. D. Modelagem Tátil, Visualização Estereoscópica e Aspectos de Avaliação em um Simulador de Coleta de Medula Óssea. In: SIMPÓSIO DE REALIDADE VIRTUAL, 2001, Florianópolis. **Anais…** Florianópolis, 2001, p. 23-31.

MACHADO, L.S.; MELLO, A.N.; ODONE F O ., V.; ZUFFO, M.K. A Virtual Reality Simulator of Pediatric Bone Marrow Harvesting Procedure. Medical Pediatric Oncology. v.39, n.4, p.282. Wiley, 2002.

MACHADO, L. S. **A Realidade Virtual no Modelamento e Simulação de Procedimentos Invasivos em Oncologia Pediátrica:** Um Estudo de Caso no Transplante de Medula Óssea. 2003. Grau: Tese (Doutorado em Engenharia Elétrica) - Escola Politécnica da Universidade de São Paulo, São Paulo, 2003.

MACHADO, L.S.; ZUFFO, M.K. Development and Evaluation of a Simulator of Invasive Procedures in Pediatric Bone Marrow Transplant. **Studies In Health Technology And Informatics**, v. 94, p. 193 – 195, 2003.

NETTO, A. V.; MACHADO, L. S.; OLIVEIRA, M .C. F. Realidade Virtual – Definições, Dispositivos e Aplicações. **Revista Eletrônica de Iniciação Científica – Publicação da Sociedade Brasileira de Computação**, v. 2, n. 1, 2002.

MICROSOFT CORPORATION. Disponível em: <http://www.microsoft.com> acesso em 2005.

MORIE, J.F. Inspiring the future: merging mass communications, art, entertainment and virtual environment. Computer Graphics, v. 28, n. 2, p. 135-138, May, 1994.

O´SULLIVAN, C.; DINGLIANA, J.; GANOVELLI, F.; BRADSHAW, G. Collision Handling for Virtual Environments. In: EUROGRAPHICS TUTORIAL, 2001. **Proceedings…** 2001.

RIQUELME, Fernando. **Estudo comparativo de soluções para detecção de colisão em tecnologias de Realidade Virtual para o desenvolvimento de ferramentas para treinamento médico**. 2005. 86 f. Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

SOURIN, A.; SOURINA, O.; TET SEN, H. Virtual Orthopedic Surgery Training. IEEE Computer Graphics and Applications, v. 20, n. 3, maio/junho de 2000.

SUN MICROSYSTEMS. The Java 3DTM API Specification Version 1.3, June 2002. Disponivel em <http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_3_API/j3dguide> acesso em 2005.

SUN MICROSYSTEMS. Tutorials & Code Camps: Java 3D API Tutorial. Disponivel em <http://developer.java.sun.com/developer/onlineTraining/java3d/> acesso em 2005.

TENDICK, F; DOWNES M.; GOKTEKIN T.; CAVUSOGLU M.C.; FEYGIN D.; WU X.; EYAL R.; HEGARTY M.; WAY L. W. A Virtual Environment Testbed for Training Laparoscopic Surgical Skills. **Presence**, v. 9, n. 3, p. 236-255, 2000.

TSAI M.D.; JOU S.B.; HSIEH M.S. An Orthopedic Virtual Reality Surgical Simulator. In: INTERNATIONAL CONFERENCE ON ARTIFICAL REALITY AND TELE-EXISTENCE (ICAT), 10, 2000, Taiwan. **Proceedings…** Taiwan, 2000.

YAGEL, R.; STREDNEY, D.; WIET, G.; SCHMALBROCK, P.; ROSENBERG, L.; SESSANNA, D.; KURZION, Y. Building a Virtual Environment for Endoscopic Sinus Surgery Simulation. Computers & Graphics, v. 20, n. 6, p. 813-823, 1996.

APENDICE A – CÓDIGOS FONTE BOX.JAVA

```
import javax.media.j3d.*;
import javax.vecmath.*;

public class Box extends Shape3D {

    public Box(double xsize, double ysize, double zsize) {
        super();
        double xmin = -xsize/2.0;
        double xmax =  xsize/2.0;
        double ymin = -ysize/2.0;
        double ymax =  ysize/2.0;
        double zmin = -zsize/2.0;
        double zmax =  zsize/2.0;

        QuadArray box = new QuadArray(24, QuadArray.COORDINATES);

        Point3d verts[] = new Point3d[24];

        box.setCapability(QuadArray.ALLOW_FORMAT_READ);
        //box.setCapability(QuadArray.ALL)

        // face frontal
        verts[0] = new Point3d(xmax, ymin, zmax);
        verts[1] = new Point3d(xmax, ymax, zmax);
        verts[2] = new Point3d(xmin, ymax, zmax);
        verts[3] = new Point3d(xmin, ymin, zmax);
        // face de tras
        verts[4] = new Point3d(xmin, ymin, zmin);
        verts[5] = new Point3d(xmin, ymax, zmin);
        verts[6] = new Point3d(xmax, ymax, zmin);
        verts[7] = new Point3d(xmax, ymin, zmin);
        // face da direita
        verts[8] = new Point3d(xmax, ymin, zmin);
        verts[9] = new Point3d(xmax, ymax, zmin);
        verts[10] = new Point3d(xmax, ymax, zmax);
        verts[11] = new Point3d(xmax, ymin, zmax);
        // face da esquerda
        verts[12] = new Point3d(xmin, ymin, zmax);
        verts[13] = new Point3d(xmin, ymax, zmax);
        verts[14] = new Point3d(xmin, ymax, zmin);
        verts[15] = new Point3d(xmin, ymin, zmin);
        // face de cima
        verts[16] = new Point3d(xmax, ymax, zmax);
        verts[17] = new Point3d(xmax, ymax, zmin);
        verts[18] = new Point3d(xmin, ymax, zmin);
        verts[19] = new Point3d(xmin, ymax, zmax);
        // face de baixo
        verts[20] = new Point3d(xmin, ymin, zmax);
        verts[21] = new Point3d(xmin, ymin, zmin);
        verts[22] = new Point3d(xmax, ymin, zmin);
        verts[23] = new Point3d(xmax, ymin, zmax);

        box.setCoordinates(0, verts);
         setGeometry(box);
        setAppearance(new Appearance());

         //ativar a area de colisao
         this.setCollidable(true);
         BoundingBox bounds = new BoundingBox(new Point3d(0.0,0.0,0.0), new Point3d(1,1,1));
         this.setCollisionBounds(bounds);
    }
}
```

APENDICE B – CÓDIGOS FONTE BOX1.JAVA

```
import javax.media.j3d.Appearance;
import javax.media.j3d.LineArray;
import javax.media.j3d.Shape3D;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;

/*
 * Created on 23/03/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Marcello Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class Box1 extends Shape3D{

	public Box1(double xSize, double ySize, double zSize){
		super();
		double xmin = -xSize/2.0;
		double xmax =  xSize/2.0;
		double ymin = -ySize/2.0;
		double ymax =  ySize/2.0;
		double zmin = -zSize/2.0;
		double zmax =  zSize/2.0;

		LineArray box = new LineArray(24, LineArray.COORDINATES | LineArray.COLOR_3);

		Point3d verts[] = new Point3d[24];

		verts[0] = new Point3d(xmax,ymax,zmax);
		verts[1] = new Point3d(xmax,ymin,zmax);

		verts[2] = new Point3d(xmax,ymax,zmin);
		verts[3] = new Point3d(xmax,ymin,zmin);

		verts[4] = new Point3d(xmin,ymax,zmin);
		verts[5] = new Point3d(xmin,ymin,zmin);

		verts[6] = new Point3d(xmin,ymax,zmax);
		verts[7] = new Point3d(xmin,ymin,zmax);

		verts[8] = new Point3d(xmax,ymax,zmax);
		verts[9] = new Point3d(xmax,ymax,zmin);

		verts[10] = new Point3d(xmax,ymax,zmin);
		verts[11] = new Point3d(xmin,ymax,zmin);

		verts[12] = new Point3d(xmin,ymax,zmin);
		verts[13] = new Point3d(xmin,ymax,zmax);

		verts[14] = new Point3d(xmin,ymax,zmax);
		verts[15] = new Point3d(xmax,ymax,zmax);

		verts[16] = new Point3d(xmax,ymin,zmax);
		verts[17] = new Point3d(xmax,ymin,zmin);

		verts[18] = new Point3d(xmax,ymin,zmin);
		verts[19] = new Point3d(xmin,ymin,zmin);

		verts[20] = new Point3d(xmin,ymin,zmin);
```

```
            verts[21] = new Point3d(xmin,ymin,zmax);

            verts[22] = new Point3d(xmin,ymin,zmax);
            verts[23] = new Point3d(xmax,ymin,zmax);

            Color3f color = new Color3f(0.5f, 0.8f, 0.5f);

            box.setCoordinates(0, verts);
            for(int i=0;i<24;i++)
                    box.setColor(i,color);
        setGeometry(box);
            setAppearance(new Appearance());
    }
}
```

APENDICE C – CÓDIGOS FONTE BBOXDETECTION.JAVA

```
import java.awt.Container;
import javax.media.j3d.Appearance;
import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.TransparencyAttributes;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;

/*
 * Created on 25/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class BBoxDetection extends JFrame {

	Box box = new Box(2,2,2);
	Sphere sphere = new Sphere();

	JTextArea textArea = new JTextArea();
	JScrollPane scrollPane;

	BoundingBox bBox = new BoundingBox();

	Container c;

	public BBoxDetection(){
		setSize(700,550);
		c = getContentPane();
		c.setLayout(null);

		Canvas3D canvas3D = new Canvas3D(null);
		c.add(canvas3D);

		c.add(scrollPane = new JScrollPane(textArea));

		canvas3D.setBounds(10,10,670,400);
	 scrollPane.setBounds(10,415,670,100);

	 SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

		BranchGroup scene = CreateSceneGraph(simpleU);
```

```
            simpleU.getViewingPlatform().setNominalViewingTransform();
         simpleU.addBranchGraph(scene);
         show();
        }

        public BranchGroup CreateSceneGraph(SimpleUniverse su){

                BranchGroup objRoot = new BranchGroup();
                TransformGroup objBox = new TransformGroup();
                TransformGroup objSphere = new TransformGroup();

                Transform3D T3DBox = new Transform3D();
                Transform3D T3DSphere = new Transform3D();

                SetCapabilityTG(objBox);
                SetCapabilityTG(objSphere);

                SetCapabilityBox(box);

                //translação do BOX
                Vector3f vectorBox = new Vector3f(5.0f,0.0f,-15.0f);
                T3DBox.setTranslation(vectorBox);
                objBox.setTransform(T3DBox);

                //translação da esfera
                Vector3f vectorSphere = new Vector3f(-5.0f,0.0f,-15.0f);
                T3DSphere.setTranslation(vectorSphere);
                objSphere.setTransform(T3DSphere);

                //Colorindo BOX
                Appearance appearanceBox = new Appearance();
                appearanceBox.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_READ);
                appearanceBox.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
                ColoringAttributes coloringAttributesBox = new ColoringAttributes();
                coloringAttributesBox.setColor(0.3f, 0.5f, 0.9f);
                TransparencyAttributes tranparencyAtt = new TransparencyAttributes(0,0.5f);
                appearanceBox.setTransparencyAttributes(tranparencyAtt);
                appearanceBox.setColoringAttributes(coloringAttributesBox);
                box.setAppearance(appearanceBox);

                //Colorindo Esfera
                Appearance appearanceSphere = new Appearance();
                appearanceSphere.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_READ);
                appearanceSphere.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
                ColoringAttributes coloringAttributesSphere = new ColoringAttributes();
                coloringAttributesSphere.setColor(0.9f, 0.5f, 0.3f);
                appearanceSphere.setColoringAttributes(coloringAttributesSphere);
                sphere.setAppearance(appearanceSphere);

                //colocando Colisão no objeto
                box.setCollisionBounds(box.getBounds());
                objBox.setBounds(box.getBounds());
                objBox.setCollisionBounds(objBox.getBounds());
                CollisionDetector collisionDetector = new CollisionDetector(box);
                collisionDetector.setSchedulingBounds(box.getBounds());
                objRoot.addChild(collisionDetector);

                sphere.setCollisionBounds(sphere.getBounds());

                System.out.println("Area da Esfera --> "+sphere.getBounds());
                System.out.println("Area do Envolvimento da Esfera --
>"+sphere.getCollisionBounds());

                objBox.addChild(box);
                objSphere.addChild(sphere);

                objRoot.addChild(objBox);
                objRoot.addChild(objSphere);

                MouseListener(objRoot,objSphere);
                KeyboardEvent(su,objRoot);
```

```
        objRoot.compile();
 return objRoot;
}

public void SetCapabilityBox(Box box){
        box.setCapability(Box.ALLOW_AUTO_COMPUTE_BOUNDS_READ);
        box.setCapability(Box.ALLOW_AUTO_COMPUTE_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_BOUNDS_READ);
        box.setCapability(Box.ALLOW_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_COLLIDABLE_READ);
        box.setCapability(Box.ALLOW_COLLIDABLE_WRITE);
        box.setCapability(Box.ALLOW_COLLISION_BOUNDS_READ);
        box.setCapability(Box.ALLOW_COLLISION_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_LOCAL_TO_VWORLD_READ);
        box.setCapability(Box.ALLOW_PICKABLE_READ);
        box.setCapability(Box.ALLOW_PICKABLE_WRITE);
}

public void SetCapabilityTG(TransformGroup tg){
        tg.setCapability(TransformGroup.ALLOW_AUTO_COMPUTE_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_AUTO_COMPUTE_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
        tg.setCapability(TransformGroup.ALLOW_COLLIDABLE_READ);
        tg.setCapability(TransformGroup.ALLOW_COLLIDABLE_WRITE);
        tg.setCapability(TransformGroup.ALLOW_COLLISION_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_COLLISION_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_LOCAL_TO_VWORLD_READ);
        tg.setCapability(TransformGroup.ALLOW_PICKABLE_READ);
        tg.setCapability(TransformGroup.ALLOW_PICKABLE_WRITE);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
}

public void MouseListener(BranchGroup bg,TransformGroup tg){

        MouseRotate myMouseRotate = new MouseRotate();
 myMouseRotate.setTransformGroup(tg);
 myMouseRotate.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseRotate);

 MouseTranslate myMouseTranslate = new MouseTranslate();
 myMouseTranslate.setTransformGroup(tg);
 myMouseTranslate.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseTranslate);

 MouseZoom myMouseZoom = new MouseZoom();
 myMouseZoom.setTransformGroup(tg);
 myMouseZoom.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseZoom);
}

public void KeyboardEvent(SimpleUniverse su,BranchGroup bg){
        Transform3D T3D = new Transform3D();

        TransformGroup vpTrans = null;

    Vector3f translate = new Vector3f(0.0f,-0.0f,0.0f);

    vpTrans = su.getViewingPlatform().getViewPlatformTransform();
    T3D.setTranslation(translate);
    vpTrans.setTransform(T3D);
    KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTrans);
    keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
    bg.addChild(keyNavBeh);

}
```

```
	public void InsertText(String text){
		textArea.append(text);
		textArea.append("\n");
	}

	public static void main(String[] args) {
		new BBoxDetection();
	}
}
```

APENDICE D – CÓDIGOS FONTE BSPHEREDETECTION.JAVA

```
import java.awt.Container;

import javax.media.j3d.Appearance;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.TransparencyAttributes;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.geometry.Sphere;
import com.sun.j3d.utils.universe.SimpleUniverse;

/*
 * Created on 09/06/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class BSphereDetection extends JFrame{

               Box box = new Box(2,2,2);
               Sphere sphere = new Sphere();

               JTextArea textArea = new JTextArea();
               JScrollPane scrollPane;

               BoundingSphere bSphere = new BoundingSphere();
               Point3d center = new Point3d();
               double radius;


               Container c;

               public BSphereDetection(){
                       setSize(700,550);
                       c = getContentPane();
                       c.setLayout(null);

                       Canvas3D canvas3D = new Canvas3D(null);
                       c.add(canvas3D);

                       c.add(scrollPane = new JScrollPane(textArea));

                       canvas3D.setBounds(10,10,670,400);
                scrollPane.setBounds(10,415,670,100);

                SimpleUniverse simpleU = new SimpleUniverse(canvas3D);
```

```
        BranchGroup scene = CreateSceneGraph(simpleU);

        simpleU.getViewingPlatform().setNominalViewingTransform();
     simpleU.addBranchGraph(scene);
     show();
    }

    public BranchGroup CreateSceneGraph(SimpleUniverse su){

        BranchGroup objRoot = new BranchGroup();
        TransformGroup objBox = new TransformGroup();
        TransformGroup objSphere = new TransformGroup();

        Transform3D T3DBox = new Transform3D();
        Transform3D T3DSphere = new Transform3D();

        SetCapabilityTG(objBox);
        SetCapabilityTG(objSphere);

        SetCapabilityBox(box);

        //translação do BOX
        Vector3f vectorBox = new Vector3f(5.0f,0.0f,-15.0f);
        T3DBox.setTranslation(vectorBox);
        objBox.setTransform(T3DBox);

        //translação da esfera
        Vector3f vectorSphere = new Vector3f(-5.0f,0.0f,-15.0f);
        T3DSphere.setTranslation(vectorSphere);
        objSphere.setTransform(T3DSphere);

        //Colorindo BOX
        Appearance appearanceBox = new Appearance();
        appearanceBox.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_READ);
        appearanceBox.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
        ColoringAttributes coloringAttributesBox = new ColoringAttributes();
        coloringAttributesBox.setColor(0.3f, 0.5f, 0.9f);
        TransparencyAttributes tranparencyAtt = new TransparencyAttributes(0,0.5f);
        appearanceBox.setTransparencyAttributes(tranparencyAtt);
        appearanceBox.setColoringAttributes(coloringAttributesBox);
        box.setAppearance(appearanceBox);

        //Colorindo Esfera
        Appearance appearanceSphere = new Appearance();
        appearanceSphere.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_READ);
        appearanceSphere.setCapability(Appearance.ALLOW_COLORING_ATTRIBUTES_WRITE);
        ColoringAttributes coloringAttributesSphere = new ColoringAttributes();
        coloringAttributesSphere.setColor(0.9f, 0.5f, 0.3f);
        appearanceSphere.setColoringAttributes(coloringAttributesSphere);
        sphere.setAppearance(appearanceSphere);

        //Colocanco Colisão para o box
        box.setCollisionBounds(box.getBounds());
        objBox.setBounds(box.getBounds());
        objBox.setCollisionBounds(objBox.getBounds());
        CollisionDetector collisionDetector = new CollisionDetector(box);
        collisionDetector.setSchedulingBounds(box.getBounds());
        objRoot.addChild(collisionDetector);

        sphere.setCollisionBounds(sphere.getBounds());

        System.out.println("Area da Esfera --> "+sphere.getBounds());
        System.out.println("Area do Envolvimento da Esfera --
>"+sphere.getCollisionBounds());

        objBox.addChild(box);
        objSphere.addChild(sphere);

        objRoot.addChild(objBox);
        objRoot.addChild(objSphere);
```

```
        MouseListener(objRoot,objSphere);
        KeyboardEvent(su,objRoot);

        bSphere.set(box.getCollisionBounds());

        bSphere.getCenter(center);
        radius = bSphere.getRadius();

        System.out.println("Center - "+center);
        System.out.println("Radius - "+radius);

 objRoot.compile();
 return objRoot;
}

public void SetCapabilityBox(Box box){
        box.setCapability(Box.ALLOW_AUTO_COMPUTE_BOUNDS_READ);
        box.setCapability(Box.ALLOW_AUTO_COMPUTE_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_BOUNDS_READ);
        box.setCapability(Box.ALLOW_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_COLLIDABLE_READ);
        box.setCapability(Box.ALLOW_COLLIDABLE_WRITE);
        box.setCapability(Box.ALLOW_COLLISION_BOUNDS_READ);
        box.setCapability(Box.ALLOW_COLLISION_BOUNDS_WRITE);
        box.setCapability(Box.ALLOW_LOCAL_TO_VWORLD_READ);
        box.setCapability(Box.ALLOW_PICKABLE_READ);
        box.setCapability(Box.ALLOW_PICKABLE_WRITE);
}

public void SetCapabilityTG(TransformGroup tg){
        tg.setCapability(TransformGroup.ALLOW_AUTO_COMPUTE_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_AUTO_COMPUTE_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_EXTEND);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_READ);
        tg.setCapability(TransformGroup.ALLOW_CHILDREN_WRITE);
        tg.setCapability(TransformGroup.ALLOW_COLLIDABLE_READ);
        tg.setCapability(TransformGroup.ALLOW_COLLIDABLE_WRITE);
        tg.setCapability(TransformGroup.ALLOW_COLLISION_BOUNDS_READ);
        tg.setCapability(TransformGroup.ALLOW_COLLISION_BOUNDS_WRITE);
        tg.setCapability(TransformGroup.ALLOW_LOCAL_TO_VWORLD_READ);
        tg.setCapability(TransformGroup.ALLOW_PICKABLE_READ);
        tg.setCapability(TransformGroup.ALLOW_PICKABLE_WRITE);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
        tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
}

public void MouseListener(BranchGroup bg,TransformGroup tg){

        MouseRotate myMouseRotate = new MouseRotate();
 myMouseRotate.setTransformGroup(tg);
 myMouseRotate.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseRotate);

 MouseTranslate myMouseTranslate = new MouseTranslate();
 myMouseTranslate.setTransformGroup(tg);
 myMouseTranslate.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseTranslate);

 MouseZoom myMouseZoom = new MouseZoom();
 myMouseZoom.setTransformGroup(tg);
 myMouseZoom.setSchedulingBounds(new BoundingSphere());
 bg.addChild(myMouseZoom);
}

public void KeyboardEvent(SimpleUniverse su,BranchGroup bg){
        Transform3D T3D = new Transform3D();

        TransformGroup vpTrans = null;
```

```
        Vector3f translate = new Vector3f(0.0f,-0.0f,0.0f);

        vpTrans = su.getViewingPlatform().getViewPlatformTransform();
        T3D.setTranslation(translate);
        vpTrans.setTransform(T3D);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTrans);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        bg.addChild(keyNavBeh);

    }

    public void InsertText(String text){
           textArea.append(text);
           textArea.append("\n");
    }

    public static void main(String[] args) {
           new BSphereDetection();
    }
}
```

APENDICE E – CÓDIGOS FONTE COLLISIONDETECTOR1.JAVA

```
import java.util.Enumeration;

import javax.media.j3d.Appearance;
import javax.media.j3d.Behavior;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.Shape3D;
import javax.media.j3d.WakeupOnCollisionEntry;
import javax.media.j3d.WakeupOnCollisionExit;
import javax.swing.JOptionPane;
import javax.vecmath.Color3f;

/*
 * Created on 25/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

public class CollisionDetector1 extends Behavior {
           private final Color3f highlightColor = new Color3f(0.0f, 1.0f, 0.0f);
           private final ColoringAttributes highlight = new
ColoringAttributes(highlightColor,ColoringAttributes.SHADE_GOURAUD);

           private boolean inCollision = false;
           private Shape3D shape;
              private ColoringAttributes shapeColoring;
           private Appearance shapeAppearance;

           private WakeupOnCollisionEntry wEnter;
           private WakeupOnCollisionExit wExit;


           public CollisionDetector1(Shape3D s) {
                     this.shape = s;
              //     this.shape.setBounds(obj.getBounds());
              //     this.shapeAppearance = ((Primitive) obj).getAppearance();
              //     this.shapeColoring = shapeAppearance.getColoringAttributes();
                     inCollision = false;
           }

              public void initialize() {
                     wEnter = new
WakeupOnCollisionEntry(shape,WakeupOnCollisionEntry.USE_GEOMETRY);
                     wExit = new
WakeupOnCollisionExit(shape,WakeupOnCollisionExit.USE_GEOMETRY);
                     wakeupOn(wEnter);
           }

           public void processStimulus(Enumeration criteria) {
                     inCollision = !inCollision;

                     if (inCollision) {
                     //     shapeAppearance.setColoringAttributes(highlight);

       //JOptionPane.showMessageDialog(null,"COLISÃO","inCollision",JOptionPane.INFORMATION_MESSA
GE);
                              System.out.println("Colisão");
                              wakeupOn(wExit);
```

```
                        }
                        else {
                        //      shapeAppearance.setColoringAttributes(shapeColoring);
                                //JOptionPane.showMessageDialog(null,"NÃO
COLISÃO","!inCollision",JOptionPane.INFORMATION_MESSAGE);
                                System.out.println("Não Colisão");
                                wakeupOn(wEnter);
                        }
            }
}
```

APENDICE F – CÓDIGOS FONTE CARREGAOBJ.JAVA

```
import java.awt.Container;
import javax.media.j3d.AmbientLight;
import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.Group;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;
import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.universe.SimpleUniverse;

/*
 * Created on 15/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class CarregaObj extends JFrame {

        ObjectFile loader = new ObjectFile();
        JTextArea textArea = new JTextArea();
        JScrollPane scrollPane;
        Container c;
        Point3d lower = new Point3d();
        Point3d upper = new Point3d();
    BoundingBox bBox = new BoundingBox();
    Shape3D s, s1;

public BranchGroup createSceneGraph(SimpleUniverse su) {
        BranchGroup objRoot = new BranchGroup();
    Transform3D T3D = new Transform3D();

    TransformGroup objRotate = new TransformGroup();
    TransformGroup objBox = new TransformGroup();

        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

    objBox.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objBox.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
```

```
objRoot.addChild(objRotate);
objRoot.addChild(objBox);

    Group grupo = carregaObj();
    Group grupo1 = carregaObj1();

    objRotate.addChild(grupo);

    objBox.addChild(grupo1);

    Vector3f transBox = new Vector3f(-2.0f, 0.0f, -5.0f);
T3D.setTranslation(transBox);
objBox.setTransform(T3D);

Vector3f transObj = new Vector3f(2.0f, 0.0f, -5.0f);
T3D.setTranslation(transObj);
objRotate.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

// Inserindo Luz ambiente
           Color3f ambientColor = new Color3f(1.0f, 1.0f, 1.0f);
           AmbientLight ambientLightNode = new AmbientLight(ambientColor);
           ambientLightNode.setInfluencingBounds(bounds);
           objRotate.addChild(ambientLightNode);

           AmbientLight ambientLightNode1 = new AmbientLight(ambientColor);
           ambientLightNode1.setInfluencingBounds(bounds);
           objBox.addChild(ambientLightNode1);

    // Inserindo Luz direcional
           Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
           Vector3f light1Direction  = new Vector3f(1.0f, 1.0f, 1.0f);
           Color3f light2Color = new Color3f(1.0f, 1.0f, 1.0f);
           Vector3f light2Direction  = new Vector3f(-1.0f, -1.0f, -1.0f);

           DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);
           light1.setInfluencingBounds(bounds);
           objRotate.addChild(light1);

           DirectionalLight light2 = new DirectionalLight(light2Color, light2Direction);
           light2.setInfluencingBounds(bounds);
           objRotate.addChild(light2);

        MouseRotate myMouseRotate = new MouseRotate();
    myMouseRotate.setTransformGroup(objBox);
    myMouseRotate.setSchedulingBounds(new BoundingSphere());
    objRoot.addChild(myMouseRotate);

    MouseTranslate myMouseTranslate = new MouseTranslate();
    myMouseTranslate.setTransformGroup(objBox);
    myMouseTranslate.setSchedulingBounds(new BoundingSphere());
    objRoot.addChild(myMouseTranslate);

    MouseZoom myMouseZoom = new MouseZoom();
    myMouseZoom.setTransformGroup(objBox);
    myMouseZoom.setSchedulingBounds(new BoundingSphere());
    objRoot.addChild(myMouseZoom);

//cria base e aciona o teclado
    TransformGroup vpTrans = null;
    Vector3f translate = new Vector3f(0.0f,-0.0f,0.0f);
    vpTrans = su.getViewingPlatform().getViewPlatformTransform();
    T3D.setTranslation(translate);
    vpTrans.setTransform(T3D);
    KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTrans);
    keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
    objRoot.addChild(keyNavBeh);

    FramesporSegundo fps = new FramesporSegundo();
           fps.setSchedulingBounds(bounds);
```

```
                objRoot.addChild(fps);

            objRoot.compile();
            return objRoot;
        }

    private Group carregaObj(){
            Scene objeto = null;
            ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);

            try
            {
                    objeto = objFileloader.load("mama.obj");
            }
            catch (Exception e)
            {
                    objeto = null;
                    System.err.println(e);
                    System.out.println("OBJ não encontrado");
            }

            BranchGroup branchGroup = objeto.getSceneGroup();
            TransformGroup objTrans = new TransformGroup();
            branchGroup.addChild(objTrans);
            Shape3D shapeObj = null;
            objTrans.addChild(shapeObj);

            shapeObj = (Shape3D) branchGroup.getChild(0);
            shapeObj.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
            shapeObj.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
            shapeObj.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
            shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_FORMAT_READ);
            shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
            shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_REF_DATA_READ);
            shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COORDINATE_READ);
            shapeObj.getGeometry().setCapability(TransformGroup.ALLOW_BOUNDS_READ);
            shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_READ);
            shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

            //coloca obj na colisão
            CollisionDetector1 cd = new CollisionDetector1(shapeObj);
            //BoundingSphere bounds = new BoundingSphere();
            BoundingBox bounds = new BoundingBox();
            bounds.set(shapeObj.getBounds());
            shapeObj.setCollisionBounds(bounds);
            System.out.println(shapeObj.getBounds());
            System.out.println(shapeObj.getCollisionBounds());
            cd.setSchedulingBounds(bounds);
            objTrans.addChild(cd);
      return branchGroup;
    }

    private Group carregaObj1(){

            Scene objeto = null;
            ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);

            try
            {
                    objeto = objFileloader.load("agulha.obj");
            }
            catch (Exception e)
            {
                    objeto = null;
                    System.err.println(e);
                    System.out.println("OBJ não encontrado");
            }


            BranchGroup branchGroup = objeto.getSceneGroup();
            TransformGroup objTrans = new TransformGroup();
```

```
        branchGroup.addChild(objTrans);
        Shape3D shapeObj = null;
        objTrans.addChild(shapeObj);

        shapeObj = (Shape3D) branchGroup.getChild(0);
        shapeObj.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
        shapeObj.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
        shapeObj.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_FORMAT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_REF_DATA_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COORDINATE_READ);
        shapeObj.getGeometry().setCapability(TransformGroup.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

  return branchGroup;
}

//////////////////////////////////////////////////////////////////////////
    public CarregaObj() {
        setSize(700,550);
                c = getContentPane();
                c.setLayout(null);

                Canvas3D canvas3D = new Canvas3D(null);
                c.add(canvas3D);

                c.add(scrollPane = new JScrollPane(textArea));

                canvas3D.setBounds(10,10,670,400);
         scrollPane.setBounds(10,415,670,100);

         SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

                BranchGroup scene = createSceneGraph(simpleU);

                simpleU.getViewingPlatform().setNominalViewingTransform();
         simpleU.addBranchGraph(scene);
         show();
    }

//////////////////////////////////////////////////////////////////////////
    public static void main(String[] args) {
        new CarregaObj();
    }

}
```

APENDICE G – CÓDIGOS FONTE CARREGAOBJ2.JAVA

```
import java.awt.GraphicsConfiguration;

import javax.media.j3d.AmbientLight;
import javax.media.j3d.Background;
import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.Group;
import javax.media.j3d.Locale;
import javax.media.j3d.PhysicalBody;
import javax.media.j3d.PhysicalEnvironment;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.media.j3d.View;
import javax.media.j3d.ViewPlatform;
import javax.media.j3d.VirtualUniverse;
import javax.swing.JFrame;
import javax.vecmath.AxisAngle4d;
import javax.vecmath.Color3f;
import javax.vecmath.Vector3d;
import javax.vecmath.Vector3f;
import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.utils.universe.SimpleUniverse;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;

/*
 * Created on 15/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to Window -
 * Preferences - Java - Code Style - Code Templates
 */
public class CarregaObj2 extends JFrame {

        ObjectFile loader = new ObjectFile();

        protected Canvas3D canvas = null;

        protected VirtualUniverse simpleU;

        protected Locale locale;

        protected BranchGroup raizMama;

        protected BranchGroup raizSeringa;

        public void setup_universo() {
                simpleU = new VirtualUniverse();
                locale = new Locale(simpleU);
                locale.addBranchGraph(buildViewBranch(canvas));
                BranchGroup raizMama = this.setup_mama();
                BranchGroup raizSeringa = this.setup_seringa();
```

```
        locale.addBranchGraph(raizMama);
        locale.addBranchGraph(raizSeringa);
}

public BranchGroup setup_mama() {

        // Cria a raiz para o gráfico "ramo"
        BranchGroup sceneRoot = new BranchGroup();

        // Crie direção luzes e fundo
        BoundingBox bounds = new BoundingBox();

        // Montando a cor de fundo
        Color3f bgColor = new Color3f(0.05f, 0.05f, 0.2f);
        Background bgNode = new Background(bgColor);
        bgNode.setApplicationBounds(bounds);
        sceneRoot.addChild(bgNode);

        // Montando as luzes globais
        // primeiro, define a cor das luzes
        Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
        Vector3f light1Direction = new Vector3f(4.0f, -7.0f, -12.0f);
        Color3f light2Color = new Color3f(0.3f, 0.3f, 0.4f);
        Vector3f light2Direction = new Vector3f(-6.0f, -2.0f, -1.0f);
        Color3f ambientColor = new Color3f(0.1f, 0.1f, 0.1f);

        // Segundo, define a luz do ambiente, e agrega ao "ramo"
        AmbientLight ambientLightNode = new AmbientLight(ambientColor);
        ambientLightNode.setInfluencingBounds(bounds);
        sceneRoot.addChild(ambientLightNode);

        // Por último, defina e agrega as luzes direcionais
        DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);
        light1.setInfluencingBounds(bounds);
        sceneRoot.addChild(light1);

        DirectionalLight light2 = new DirectionalLight(light2Color,light2Direction);
        light2.setInfluencingBounds(bounds);
        sceneRoot.addChild(light2);

        Group mama = carregaMama();
        TransformGroup tgmama = new TransformGroup();
        tgmama.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        Transform3D posmama = new Transform3D();
        // posmama.setScale(0.7);
        //posmama.setRotation(new AxisAngle4d(1.0, 1.0, 1.0, 0.778));
        posmama.setTranslation(new Vector3f(0.5f, 0.0f, 0.0f));
        tgmama.setTransform(posmama);
        tgmama.addChild(mama);

        sceneRoot.addChild(tgmama);
        sceneRoot.compile();

        return sceneRoot;

}

public BranchGroup setup_seringa() {

        BranchGroup sceneRoot = new BranchGroup();
        BoundingBox bounds = new BoundingBox();

                TransformGroup transpick = new TransformGroup();
                transpick.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
                transpick.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
                transpick.setCapability(TransformGroup.ENABLE_PICK_REPORTING);
                sceneRoot.addChild(transpick);

                TransformGroup objTG = new TransformGroup();
                Transform3D objTrans = new Transform3D();
                objTG.getTransform(objTrans);
```

```
                transpick.addChild(objTG);

                Group agulha = carregaAgulha();
                TransformGroup tgagulha = new TransformGroup();
                tgagulha.addChild(agulha);
                Transform3D t3dagulha = new Transform3D();
                t3dagulha.setTranslation(new Vector3d(0.5f, -0.2f, 0.055f));
                t3dagulha.setScale(0.2);
                tgagulha.setTransform(t3dagulha);
                objTG.addChild(tgagulha);

                Transform3D posagulha = new Transform3D();
                posagulha.setScale(0.5);
                posagulha.setTranslation(new Vector3f(-0.5f, 0.0f, 0.0f));
                tgagulha.setTransform(posagulha);


        MouseRotate myMouseRotate = new MouseRotate();
        myMouseRotate.setTransformGroup(transpick);
        myMouseRotate.setSchedulingBounds(new BoundingSphere());
        sceneRoot.addChild(myMouseRotate);

        MouseTranslate myMouseTranslate = new MouseTranslate();
        myMouseTranslate.setTransformGroup(transpick);
        myMouseTranslate.setSchedulingBounds(new BoundingSphere());
        sceneRoot.addChild(myMouseTranslate);

        MouseZoom myMouseZoom = new MouseZoom();
        myMouseZoom.setTransformGroup(transpick);
        myMouseZoom.setSchedulingBounds(new BoundingSphere());
        sceneRoot.addChild(myMouseZoom);


        FramesporSegundo fps = new FramesporSegundo();
        fps.setSchedulingBounds(bounds);
        sceneRoot.addChild(fps);

        sceneRoot.compile();

        return sceneRoot;

}

private Group carregaMama() {

        Scene mama = null;
        ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);

        try {
                mama = objFileloader.load("mama.obj");
        } catch (Exception e) {
                mama = null;
                System.err.println(e);
        }

        BranchGroup branchGroup = mama.getSceneGroup();
        TransformGroup objTrans = new TransformGroup();
        branchGroup.addChild(objTrans);
        Shape3D shapeMama = null;
        objTrans.addChild(shapeMama);

        /*
         * Shape3D shapeMama0 = (Shape3D) branchGroup.getChild(0);
         * shapeMama0.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
         *
         * TransformGroup objTrans = new TransformGroup();
         * branchGroup.addChild(objTrans); objTrans.addChild(shapeMama0);
         */

        shapeMama = (Shape3D) branchGroup.getChild(0);
        shapeMama.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
```

```
        shapeMama.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
        shapeMama.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
        shapeMama.setCapability(Shape3D.ALLOW_BOUNDS_READ);
        shapeMama.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

        shapeMama.getGeometry().setCapability(GeometryArray.ALLOW_FORMAT_READ);
        shapeMama.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
        shapeMama.getGeometry()
                        .setCapability(GeometryArray.ALLOW_REF_DATA_READ);

        CollisionDetector cd = new CollisionDetector(shapeMama);
        BoundingBox bounds = new BoundingBox();
        cd.setSchedulingBounds(bounds);
        objTrans.addChild(cd);

        return branchGroup;

}

private Group carregaAgulha() {
        Scene agulha = null;
        ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);

        try {
                agulha = objFileloader.load("agulha.obj");
        } catch (Exception e) {
                agulha = null;
                System.err.println(e);
        }

        BranchGroup branchGroup = agulha.getSceneGroup();
        TransformGroup objTrans = new TransformGroup();
        branchGroup.addChild(objTrans);
        Shape3D shapeAgulha = null;
        objTrans.addChild(shapeAgulha);

        shapeAgulha = (Shape3D) branchGroup.getChild(0);
        shapeAgulha.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
        shapeAgulha.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
        shapeAgulha.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
        shapeAgulha.setCapability(Shape3D.ALLOW_BOUNDS_READ);
        shapeAgulha.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

        shapeAgulha.getGeometry()
                        .setCapability(GeometryArray.ALLOW_FORMAT_READ);
        shapeAgulha.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
        shapeAgulha.getGeometry().setCapability(
                        GeometryArray.ALLOW_REF_DATA_READ);
        shapeAgulha.getGeometry().setCapability(
                        GeometryArray.ALLOW_COORDINATE_READ);

        /*
         * CollisionDetector cd = new CollisionDetector(shapeAgulha);
         * BoundingBox bounds = new BoundingBox ();
         * cd.setSchedulingBounds(bounds); objTrans.addChild(cd);
         */

        return branchGroup;

}

protected BranchGroup buildViewBranch(Canvas3D c) {
        BranchGroup viewBranch = new BranchGroup();
        Transform3D viewXfm = new Transform3D();
        viewXfm.set(new Vector3f(0.0f, 0.0f, 4.0f));
        TransformGroup viewXfmGroup = new TransformGroup(viewXfm);
        ViewPlatform myViewPlatform = new ViewPlatform();
        PhysicalBody myBody = new PhysicalBody();
        PhysicalEnvironment myEnvironment = new PhysicalEnvironment();
        viewXfmGroup.addChild(myViewPlatform);
        viewBranch.addChild(viewXfmGroup);
```

```
            View myView = new View();
            myView.addCanvas3D(c);
            myView.attachViewPlatform(myViewPlatform);
            myView.setPhysicalBody(myBody);
            myView.setPhysicalEnvironment(myEnvironment);
            return viewBranch;
        }

        // ////////////////////////////////////////////////////////////////////////
        public CarregaObj2() {
            setSize(700, 550);
            GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
            canvas = new Canvas3D(config);
            canvas.setBounds(10, 10, 670, 400);
            getContentPane().add(canvas);
            setup_universo();
            show();
        }

        // ////////////////////////////////////////////////////////////////////////
        public static void main(String[] args) {
            new CarregaObj2();
        }

}
```

APENDICE H – CÓDIGOS FONTE CARREGAOBJ3.JAVA

```
import java.awt.Container;
import javax.media.j3d.AmbientLight;
import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.BranchGroup;
import javax.media.j3d.Canvas3D;
import javax.media.j3d.DirectionalLight;
import javax.media.j3d.GeometryArray;
import javax.media.j3d.Group;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.TransformGroup;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;
import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.utils.behaviors.keyboard.KeyNavigatorBehavior;
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;
import com.sun.j3d.utils.universe.SimpleUniverse;

/*
 * Created on 15/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class CarregaObj3 extends JFrame {

        ObjectFile loader = new ObjectFile();
        JTextArea textArea = new JTextArea();
        JScrollPane scrollPane;
        Container c;
        Point3d lower = new Point3d();
        Point3d upper = new Point3d();
    BoundingBox bBox = new BoundingBox();

public BranchGroup createSceneGraph(SimpleUniverse su) {

        BranchGroup objRoot = new BranchGroup();
        Transform3D T3D = new Transform3D();

    TransformGroup objRotate = new TransformGroup();

        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

    objRoot.addChild(objRotate);

        Group grupo = carregaObj();

        objRotate.addChild(grupo);
```

```
        Vector3f transObj = new Vector3f(2.0f, 0.0f, -5.0f);
    T3D.setTranslation(transObj);
    objRotate.setTransform(T3D);

    BoundingBox bounds = new BoundingBox();

    // Inserindo Luz ambiente
              Color3f ambientColor = new Color3f(1.0f, 1.0f, 1.0f);
              AmbientLight ambientLightNode = new AmbientLight(ambientColor);
              ambientLightNode.setInfluencingBounds(bounds);
              objRotate.addChild(ambientLightNode);

              // Inserindo Luz direcional
              Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
              Vector3f light1Direction  = new Vector3f(1.0f, 1.0f, 1.0f);
              Color3f light2Color = new Color3f(1.0f, 1.0f, 1.0f);
              Vector3f light2Direction  = new Vector3f(-1.0f, -1.0f, -1.0f);

              DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);
              light1.setInfluencingBounds(bounds);
              objRotate.addChild(light1);

              DirectionalLight light2 = new DirectionalLight(light2Color, light2Direction);
              light2.setInfluencingBounds(bounds);
              objRotate.addChild(light2);


    //cria base e aciona o teclado
        TransformGroup vpTrans = null;
        Vector3f translate = new Vector3f(0.0f,-0.0f,0.0f);
        vpTrans = su.getViewingPlatform().getViewPlatformTransform();
        T3D.setTranslation(translate);
        vpTrans.setTransform(T3D);
        KeyNavigatorBehavior keyNavBeh = new KeyNavigatorBehavior(vpTrans);
        keyNavBeh.setSchedulingBounds(new BoundingSphere(new Point3d(),1000.0));
        objRoot.addChild(keyNavBeh);

        FramesporSegundo fps = new FramesporSegundo();
              fps.setSchedulingBounds(bounds);
              objRoot.addChild(fps);

        objRoot.compile();
        return objRoot;
    }


public BranchGroup createSceneGraph1(SimpleUniverse su) {

        BranchGroup objRoot = new BranchGroup();
        Transform3D T3D = new Transform3D();

    TransformGroup objRotate = new TransformGroup();

        objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRotate.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);

    objRoot.addChild(objRotate);

        Group grupo = carregaObj1();

        objRotate.addChild(grupo);

        Vector3f transObj = new Vector3f(2.0f, 0.0f, -5.0f);
    T3D.setTranslation(transObj);
    objRotate.setTransform(T3D);

    BoundingBox bounds = new BoundingBox();

    // Inserindo Luz ambiente
              Color3f ambientColor = new Color3f(1.0f, 1.0f, 1.0f);
              AmbientLight ambientLightNode = new AmbientLight(ambientColor);
```

```
                ambientLightNode.setInfluencingBounds(bounds);
                objRotate.addChild(ambientLightNode);


        // Inserindo Luz direcional
                Color3f light1Color = new Color3f(1.0f, 1.0f, 0.9f);
                Vector3f light1Direction  = new Vector3f(1.0f, 1.0f, 1.0f);
                Color3f light2Color = new Color3f(1.0f, 1.0f, 1.0f);
                Vector3f light2Direction  = new Vector3f(-1.0f, -1.0f, -1.0f);

                DirectionalLight light1 = new DirectionalLight(light1Color, light1Direction);
                light1.setInfluencingBounds(bounds);
                objRotate.addChild(light1);

                DirectionalLight light2 = new DirectionalLight(light2Color, light2Direction);
                light2.setInfluencingBounds(bounds);
                objRotate.addChild(light2);

            FramesporSegundo fps = new FramesporSegundo();
                fps.setSchedulingBounds(bounds);
                objRoot.addChild(fps);

         objRoot.compile();
         return objRoot;
    }

private Group carregaObj()
{
        Scene objeto = null;
        ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);
        Transform3D T3D = new Transform3D();

        try
        {
                objeto = objFileloader.load("mama.obj");
        }
        catch (Exception e)
        {
                objeto = null;
                System.err.println(e);
                System.out.println("OBJ não encontrado");
        }

        BranchGroup branchGroup = objeto.getSceneGroup();
        TransformGroup objTrans = new TransformGroup();
        branchGroup.addChild(objTrans);
        Shape3D shapeObj = null;
        objTrans.addChild(shapeObj);

        shapeObj = (Shape3D) branchGroup.getChild(0);
        shapeObj.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
        shapeObj.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
        shapeObj.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_FORMAT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_REF_DATA_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COORDINATE_READ);
        shapeObj.getGeometry().setCapability(TransformGroup.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

        //coloca obj na colisão
        CollisionDetector cd = new CollisionDetector(shapeObj);
        BoundingSphere bounds = new BoundingSphere();
        bounds.set(shapeObj.getBounds());
        shapeObj.setCollisionBounds(bounds);
        System.out.println(shapeObj.getBounds());
        System.out.println(shapeObj.getCollisionBounds());
        cd.setSchedulingBounds(bounds);
        objTrans.addChild(cd);
  return branchGroup;
```

```
}

private Group carregaObj1()
{
        Scene objeto = null;
        ObjectFile objFileloader = new ObjectFile(ObjectFile.RESIZE);

        try
        {
                objeto = objFileloader.load("agulha.obj");
        }
        catch (Exception e)
        {
                objeto = null;
                System.err.println(e);
                System.out.println("OBJ não encontrado");
        }


        BranchGroup branchGroup = objeto.getSceneGroup();
        TransformGroup objTrans = new TransformGroup();
        branchGroup.addChild(objTrans);
        Shape3D shapeObj = null;
        objTrans.addChild(shapeObj);

        shapeObj = (Shape3D) branchGroup.getChild(0);
        shapeObj.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
        shapeObj.setCapability(Shape3D.ALLOW_GEOMETRY_READ);
        shapeObj.setCapability(Shape3D.ALLOW_COLLISION_BOUNDS_WRITE);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_FORMAT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COUNT_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_REF_DATA_READ);
        shapeObj.getGeometry().setCapability(GeometryArray.ALLOW_COORDINATE_READ);
        shapeObj.getGeometry().setCapability(TransformGroup.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_READ);
        shapeObj.setCapability(Shape3D.ALLOW_BOUNDS_WRITE);

  return branchGroup;
}

//////////////////////////////////////////////////////////////////////////
    public CarregaObj3() {
        setSize(700,550);
                c = getContentPane();
                c.setLayout(null);

                Canvas3D canvas3D = new Canvas3D(null);
                c.add(canvas3D);

                c.add(scrollPane = new JScrollPane(textArea));

                canvas3D.setBounds(10,10,670,400);
         scrollPane.setBounds(10,415,670,100);

         SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

                BranchGroup scene = createSceneGraph(simpleU);
                BranchGroup scene1 = createSceneGraph1(simpleU);

                simpleU.getViewingPlatform().setNominalViewingTransform();
         simpleU.addBranchGraph(scene);
         simpleU.addBranchGraph(scene1);
         show();
    }

//////////////////////////////////////////////////////////////////////////
    public static void main(String[] args) {
        new CarregaObj();
    }

}
```

APENDICE I – CÓDIGOS FONTE COLLISIONDETECTOR.JAVA

```
import java.lang.reflect.Array;
import java.util.Enumeration;
import javax.media.j3d.Appearance;
import javax.media.j3d.Behavior;
import javax.media.j3d.BoundingBox;
import javax.media.j3d.BoundingSphere;
import javax.media.j3d.ColoringAttributes;
import javax.media.j3d.SceneGraphPath;
import javax.media.j3d.Shape3D;
import javax.media.j3d.Transform3D;
import javax.media.j3d.WakeupCriterion;
import javax.media.j3d.WakeupOnCollisionEntry;
import javax.media.j3d.WakeupOnCollisionExit;
import javax.media.j3d.WakeupOnCollisionMovement;
import javax.media.j3d.WakeupOr;
import javax.vecmath.Color3f;
import javax.vecmath.Point3d;
import javax.vecmath.Vector3f;

/*
 * Created on 25/02/2005
 *
 * TODO To change the template for this generated file go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

/**
 * @author Kera
 *
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */

public class CollisionDetector extends Behavior {
           private final Color3f highlightColor = new Color3f(0.0f, 1.0f, 0.0f);
           private final ColoringAttributes highlight = new
ColoringAttributes(highlightColor,ColoringAttributes.SHADE_GOURAUD);

           private boolean inCollision = false;
           private Shape3D shape;
              private ColoringAttributes shapeColoring;
           private Appearance shapeAppearance;

           private WakeupCriterion[] wCriterion;
           private WakeupOr wOr;

           private Point3d lower = new Point3d();
           private    Point3d upper = new Point3d();
              private Point3d center = new Point3d();

           private BoundingBox boundingBox = new BoundingBox();
           private BoundingSphere boundingSphere = new BoundingSphere();
           private Vector3f v3f = new Vector3f();
           private Vector3f v3f1 = new Vector3f();
           private Point3d ultimoCentro = new Point3d();

           public CollisionDetector(Shape3D shape) {
                     this.shape = shape;
                     //this.shape.setBounds(shape.getBounds());
                     boundingBox.set(shape.getBounds());
                     boundingSphere.set(shape.getBounds());

                     System.out.println(boundingBox.toString());
                     System.out.println(boundingSphere.toString());
```

```
                    boundingBox.getLower(lower);
                boundingBox.getUpper(upper);
                center.set((lower.x+upper.x)/2,(lower.y+upper.y)/2,(lower.z+upper.z)/2);

                    inCollision = false;
    }

            public void initialize() {
                wCriterion= new WakeupCriterion[3];

                wCriterion[0] = new WakeupOnCollisionEntry(shape,
WakeupOnCollisionEntry.USE_BOUNDS);
                    wCriterion[1] = new
WakeupOnCollisionExit(shape,WakeupOnCollisionExit.USE_BOUNDS);
                    wCriterion[2] = new
WakeupOnCollisionMovement(shape,WakeupOnCollisionMovement.USE_BOUNDS);
                    //wCriterion[0] = new WakeupOnCollisionEntry(shape,
WakeupOnCollisionEntry.USE_GEOMETRY);
                    //wCriterion[1] = new
WakeupOnCollisionExit(shape,WakeupOnCollisionExit.USE_GEOMETRY);
                    //wCriterion[2] = new
WakeupOnCollisionMovement(shape,WakeupOnCollisionMovement.USE_GEOMETRY);

                    wOr = new WakeupOr(wCriterion);
                    wakeupOn(wOr);
            }

            public void processStimulus(Enumeration criteria) {


                System.out.println("Upper 1 -> "+ upper);
                    System.out.println("lower 1-> "+ lower);
                    System.out.println("center 1 -> "+ center);

                    ultimoCentro = center;

                while (criteria.hasMoreElements()) {

                            Object ProximoElemento = criteria.nextElement();
                            WakeupCriterion wakeup = (WakeupCriterion) ProximoElemento;

                            if (wakeup instanceof WakeupOnCollisionEntry){

                                    System.out.println("Entrando nos limites de colisão
(USE_BOUNDS)...");
                            }
                            else{
                                    if (wakeup instanceof WakeupOnCollisionMovement){
                                            WakeupOnCollisionMovement ec =
(WakeupOnCollisionMovement) ProximoElemento;

                                            SceneGraphPath sg = ec.getArmingPath(); //Objeto
utilizado na condição de colisão
                                            Transform3D t3d = new Transform3D();
                                            sg.getObject().getLocalToVworld(t3d);

                                            t3d.get(v3f);

                                            SceneGraphPath sg1 = ec.getTriggeringPath(); //Objeto
que causou a colisão
                                            Transform3D t3d1 = new Transform3D();
                                            sg1.getObject().getLocalToVworld(t3d1);

                                            t3d1.get(v3f1);

                                            //Calcula distância euclidiana
                                            double dist = Math.sqrt(Math.pow((lower.x-
upper.x),2)+Math.pow((lower.y-upper.y),2)+Math.pow((lower.z-upper.z),2));
                                            System.out.println("Distancia Euclidiana do BBox em
relação ao objeto");
                                            System.out.println("\n"+dist);
```

```
                                        double dist1 = Math.sqrt(Math.pow((v3f.x -
v3f1.x),2)+Math.pow((v3f.y - v3f1.y),2)+Math.pow((v3f.z - v3f1.z),2));

                                        if (dist < 0.01){
                                                if(!boundingBox.isEmpty()){
                                                        System.out.println("Colisão");
                                                }
                                                else{
                                                        System.out.println("SAI");
                                                        boundingBox.setLower(lower);
                                                        boundingBox.setUpper(upper);
                                                        shape.setBounds(boundingBox);

        shape.setCollisionBounds(boundingSphere);
                                                        System.out.println("Retornando a
BBoxOriginal");
                                                }
                                        }
                                        else{
                                                if(boundingBox.isEmpty()){
                                                        System.out.println("não Colisão");
                                                        boundingBox.setLower(lower);
                                                        boundingBox.setUpper(upper);
                                                        shape.setBounds(boundingBox);
                                                        shape.setCollisionBounds(boundingBox);

                                                        System.out.println("Retornando a
BBoxOriginal");
                                                }
                                                else{
                                                        refina();
                                                        System.out.println("REFINANDO...");

                                                }
                                        }
                                }
                                else{
                                        if (wakeup instanceof WakeupOnCollisionExit){
                                                System.out.println("Fora dos limites de
colisão...");
                                        }
                                }
                        }
                }
            wakeupOn(wOr);
        }

        public void refina(){

                Point3d[] pointsLow = (Point3d[])Array.newInstance(Point3d.class,8);
                Point3d[] pointsUpp = (Point3d[])Array.newInstance(Point3d.class,8);
                Point3d[] pointsCenter = (Point3d[])Array.newInstance(Point3d.class,8);

                double[] menorDistancia = new double[8];
                double dist;

                for(int i=0;i<pointsLow.length;i++){
                        pointsLow[i] = new Point3d();
                        pointsUpp[i] = new Point3d();
                        pointsCenter[i] = new Point3d();
                }

                pointsLow[0].set(lower);
                        pointsUpp[0].set(center);

                        System.out.println("Pointslow[0]->"+pointsLow[0]);
                        System.out.println("PointsUpp[0]->"+pointsUpp[0]);


        pointsCenter[0].set((pointsLow[0].x+pointsUpp[0].x)/2,(pointsLow[0].y+pointsUpp[0].y)/2,(p
```

```
ointsLow[0].z+pointsUpp[0].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[0].x),2)+Math.pow((v3f1.y
- pointsCenter[0].y),2)+Math.pow((v3f1.z - pointsCenter[0].z),2));
                        menorDistancia[0] = dist;



                        pointsLow[1].set(center.x,lower.y,lower.z);
                        pointsUpp[1].set(upper.x,center.y,center.z);

                        System.out.println("Pointslow[1]->"+pointsLow[1]);
                        System.out.println("PointsUpp[1]->"+pointsUpp[1]);


       pointsCenter[1].set((pointsLow[1].x+pointsUpp[1].x)/2,(pointsLow[1].y+pointsUpp[1].y)/2,(p
ointsLow[1].z+pointsUpp[1].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[1].x),2)+Math.pow((v3f1.y
- pointsCenter[1].y),2)+Math.pow((v3f1.z - pointsCenter[1].z),2));
                        menorDistancia[1] = dist;



                        pointsLow[2].set(center.x,lower.y,center.z);
                        pointsUpp[2].set(upper.x,center.y,upper.z);

                        System.out.println("Pointslow[2]->"+pointsLow[2]);
                        System.out.println("PointsUpp[2]->"+pointsUpp[2]);


       pointsCenter[2].set((pointsLow[2].x+pointsUpp[2].x)/2,(pointsLow[2].y+pointsUpp[2].y)/2,(p
ointsLow[2].z+pointsUpp[2].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[2].x),2)+Math.pow((v3f1.y
- pointsCenter[2].y),2)+Math.pow((v3f1.z - pointsCenter[2].z),2));
                        menorDistancia[2] = dist;



                        pointsLow[3].set(lower.x,lower.y,center.z);
                        pointsUpp[3].set(center.x,center.y,upper.z);

                        System.out.println("Pointslow[3]->"+pointsLow[3]);
                        System.out.println("PointsUpp[3]->"+pointsUpp[3]);


       pointsCenter[3].set((pointsLow[3].x+pointsUpp[3].x)/2,(pointsLow[3].y+pointsUpp[3].y)/2,(p
ointsLow[3].z+pointsUpp[3].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[3].x),2)+Math.pow((v3f1.y
- pointsCenter[3].y),2)+Math.pow((v3f1.z - pointsCenter[3].z),2));
                        menorDistancia[3] = dist;



                        pointsLow[4].set(lower.x,center.y,lower.z);
                        pointsUpp[4].set(center.x,upper.y,center.z);

                        System.out.println("Pointslow[4]->"+pointsLow[4]);
                        System.out.println("PointsUpp[4]->"+pointsUpp[4]);


       pointsCenter[4].set((pointsLow[4].x+pointsUpp[4].x)/2,(pointsLow[4].y+pointsUpp[4].y)/2,(p
ointsLow[4].z+pointsUpp[4].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[4].x),2)+Math.pow((v3f1.y
- pointsCenter[4].y),2)+Math.pow((v3f1.z - pointsCenter[4].z),2));
                        menorDistancia[4] = dist;



                        pointsLow[5].set(center.x,center.y,lower.z);
                        pointsUpp[5].set(upper.x,upper.y,center.z);

                        System.out.println("Pointslow[5]->"+pointsLow[5]);
```

```
                        System.out.println("PointsUpp[5]->"+pointsUpp[5]);


       pointsCenter[5].set((pointsLow[5].x+pointsUpp[5].x)/2,(pointsLow[5].y+pointsUpp[5].y)/2,(p
ointsLow[5].z+pointsUpp[5].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[5].x),2)+Math.pow((v3f1.y
-pointsCenter[5].y),2)+Math.pow((v3f1.z - pointsCenter[5].z),2));
                        menorDistancia[5] = dist;



                        pointsLow[6].set(center);
                        pointsUpp[6].set(upper);

                        System.out.println("Pointslow[6]->"+pointsLow[6]);
                        System.out.println("PointsUpp[6]->"+pointsUpp[6]);


       pointsCenter[6].set((pointsLow[6].x+pointsUpp[6].x)/2,(pointsLow[6].y+pointsUpp[6].y)/2,(p
ointsLow[6].z+pointsUpp[6].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[6].x),2)+Math.pow((v3f1.y
- pointsCenter[6].y),2)+Math.pow((v3f1.z - pointsCenter[6].z),2));
                        menorDistancia[6] = dist;



                        pointsLow[7].set(lower.x,center.y,center.z);
                        pointsUpp[7].set(center.x,upper.y,upper.z);

                        System.out.println("Pointslow[7]->"+pointsLow[7]);
                        System.out.println("PointsUpp[7]->"+pointsUpp[7]);


       pointsCenter[7].set((pointsLow[7].x+pointsUpp[7].x)/2,(pointsLow[7].y+pointsUpp[7].y)/2,(p
ointsLow[7].z+pointsUpp[7].z)/2);
                        dist = Math.sqrt(Math.pow((v3f1.x - pointsCenter[7].x),2)+Math.pow((v3f1.y
- pointsCenter[7].y),2)+Math.pow((v3f1.z - pointsCenter[7].z),2));
                        menorDistancia[7] = dist;

                        double menor;
                        int indice;
                        indice = 0;
                        menor = menorDistancia[0];
                        System.out.println("Distancia[0] ->"+menorDistancia[0]);
                        for(int i = 1;i<8;i++){
                                System.out.println("Distancia["+i+"] ->"+menorDistancia[i]);
                                if (menorDistancia[i] < menor){
                                        menor = menorDistancia[i];
                                        indice = i;
                                }
                        }
                        System.out.println("Indice: "+indice);
                        System.out.println("Menor Distancia["+indice+"] ->
"+menorDistancia[indice]);
                        boundingSphere.setCenter(pointsCenter[indice]);
                        boundingSphere.setRadius(Math.sqrt(Math.pow((pointsCenter[indice].x -
ultimoCentro.x),2)+Math.pow((pointsCenter[indice].y -
ultimoCentro.y),2)+Math.pow((pointsCenter[indice].z - ultimoCentro.z),2)));
                        lower.set(pointsLow[indice]);
                        upper.set(pointsUpp[indice]);
                        center.set(pointsCenter[indice]);

                        System.out.println("Upper -> "+ upper);
                        System.out.println("lower -> "+ lower);
                        System.out.println("center -> "+ center);

                        boundingBox.setLower(lower);
                        boundingBox.setUpper(upper);
                        shape.setBounds(boundingBox);
                        shape.setCollisionBounds(boundingSphere);
           }}
```

APENDICE J – CÓDIGOS FONTE FRAMESPORSEGUNDO.JAVA

```
import java.io.File;
import java.io.RandomAccessFile;
import java.util.Enumeration;

import javax.media.j3d.Behavior;
import javax.media.j3d.WakeupCondition;
import javax.media.j3d.WakeupCriterion;
import javax.media.j3d.WakeupOnElapsedFrames;

public class FramesporSegundo extends Behavior
{

	protected WakeupCondition Condicao = null;
	protected long TempoInicial = 0;
	private final int IntervaloRelatorio = 100;
	File file = new File("FPS.fps");


	public FramesporSegundo()
	{
		this.Condicao = new WakeupOnElapsedFrames(IntervaloRelatorio);
	}


	public void initialize()
	{
		this.wakeupOn(Condicao);
	}

	public void processStimulus(Enumeration criteria)
	{
		while (criteria.hasMoreElements())
		{
			WakeupCriterion wakeUp = (WakeupCriterion) criteria.nextElement();
			if (wakeUp instanceof WakeupOnElapsedFrames)
			{
				if (TempoInicial>0)
				{
					final long Intervalo = System.currentTimeMillis() -
TempoInicial;

					try{
						RandomAccessFile r = new RandomAccessFile(file,
"rw");
						double num = (double)IntervaloRelatorio /
(Intervalo/1000.00);
						r.seek(r.length());
						r.writeBytes(num+"\n");
							}catch (Exception e){
								e.printStackTrace();
								}
					//System.out.println((double)IntervaloRelatorio /
(Intervalo/1000.00));
				}
				TempoInicial = System.currentTimeMillis();
			}
		}
		wakeupOn (Condicao);
	}
}
```