FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
PROGRAMA DE MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDO RIQUELME

ESTUDO COMPARATIVO DE SOLUÇÕES PARA DETECÇÃO DE COLISÃO EM TECNOLOGIAS DE REALIDADE VIRTUAL PARA O DESENVOLVIMENTO DE FERRAMENTAS PARA TREINAMENTO MÉDICO

MARÍLIA
2005

FERNANDO RIQUELME

ESTUDO COMPARATIVO DE SOLUÇÕES PARA DETECÇÃO DE COLISÃO EM TECNOLOGIAS DE REALIDADE VIRTUAL PARA O DESENVOLVIMENTO DE FERRAMENTAS PARA TREINAMENTO MÉDICO

Dissertação apresentada ao Programa de Mestrado do Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, para obtenção do Título de Mestre em Ciência da Computação (Área de Concentração: Realidade Virtual).

Orientadora:
Profa. Dra. Fátima de Lourdes dos Santos Nunes Marques

MARÍLIA
2005

RIQUELME, Fernando

Estudo Comparativo de Soluções para Detecção de Colisão em Tecnologias de Realidade Virtual para o Desenvolvimento de Ferramentas para Treinamento Médico / Fernando Riquelme; orientadora: Fátima de Lourdes dos Santos Nunes Marques. Marília, SP: [s.n.], 2005.

86 f.

Dissertação (Mestrado em Ciência da Computação) - Centro Universitário Eurípides de Marília - Fundação de Ensino Eurípides Soares da Rocha.

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

CDD: 006

FERNANDO RIQUELME

ESTUDO COMPARATIVO DE SOLUÇÕES PARA DETECÇÃO DE COLISÃO EM TECNOLOGIAS DE REALIDADE VIRTUAL PARA O DESENVOLVIMENTO DE FERRAMENTAS PARA TREINAMENTO MÉDICO

Banca examinadora da dissertação apresentada ao Programa de Mestrado da UNIVEM,/F.E.E.S.R., para obtenção do Título de Mestre em Ciência da Computação. Área de Concentração: Realidade Virtual.

Resultado: APROVADO.

ORIENTADORA: _____

Profa. Dra. Fátima de Lourdes dos Santos Nunes Marques

1º EXAMINADOR: _____

Prof. Dr. Alexandre Cardoso

2º EXAMINADOR: _____

Prof. Dr. Ildeberto Aparecido Rodello

Marília, 03 de fevereiro de 2005.

Dedico este trabalho a todos que, de maneira direta e indireta, deram o apoio necessário para enfrentar esta jornada.

AGRADECIMENTOS

Agradeço:

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

À minha Mãe, Dona Maria, por nunca ter desistido de lutar pelos seus filhos e ser a precursora de toda a minha jornada!

À minha esposa Ana Paula e aos meus filhos, Luiz Fernando e Luiz Gustavo, por sobreviverem às turbulências deste período!

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

Ao Valdir e Gilberto (Tesouraria – UNIVEM) pela atenção e compreensão!

Aos colegas da Instituição Toledo de Ensino que acreditam no meu profissionalismo e sempre confiaram como amigo!

À professora Maria Luiza por ser minha co-orientadora pessoal!

Ao Roberval pela prova de amizade!

Aos meus amigos e irmãos de coração: Marcos, Evandro e Ewerton, que sempre estiveram presentes!

Por fim, aos amigos dos Correios pela amizade contínua, principalmente, Ulisses, Marcos, Vladimir e André.

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.

RESUMO

A detecção de colisão é um dos fatores mais importantes para a obtenção de realismo num Ambiente Virtual. Este problema é fundamental em qualquer simulação do mundo físico, sendo estudado por diversas comunidades. Em simulação de procedimentos médicos, além da detecção da colisão, é importante saber onde ocorreu o impacto e se houve, ou não, interpenetração entre objetos. Nessas aplicações, as questões de precisão e tempo de resposta são fundamentais. Este trabalho apresenta um estudo sobre os métodos e principais abordagens oferecidas por tecnologias de *software* de Realidade Virtual para detecção de colisão. Além disso, é apresentada uma análise sobre os métodos existentes na biblioteca Java 3D e no *software* WorldToolKit, uma proposta de refinamento com maior precisão e menor custo computacional e, por fim, uma análise comparativa envolvendo os métodos estudados e implementados. Todos os testes foram realizados tendo como base um protótipo de simulador para punção de mama e uma plataforma padrão PC.

Palavras-chave: Detecção de Colisão. Realidade Virtual. Medicina.

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.

ABSTRACT

The collision detection is one of the most important factors for obtaining realism in a Virtual Environment. This is a basic problem in a simulation of the physical world, have being studied in several communities. In simulation of medical procedures, beyond the collision detection, it is important to know where the impact occurred and if it had, or not, interpenetration between objects. In these applications, the questions of precision and time of replying are extremely important. This work presents a study on the methods and main approaches offered by technologies of software of Virtual Reality for collision detection. Moreover, is presented an analysis on the existing methods in the Java 3D API and software WorldToolKit, a proposal of refinement with greater precision and minor computational cost and, finally, a comparative analysis involving the studied and implemented methods. All the tests had been made considering a tool that is a prototype of a simulator for executing the core biopsy exam by using a platform standard PC.

Keywords: Collision Detection. Virtual Reality. Medicine.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

2D – Bidimensional

3D – Tridimensional

AV – Ambiente Virtual

CAD – *Computer Aided Design*

DOF – *Degrees of Freedom*

FLTK – *Fast Light ToolKit*

FPS – *Frames* por Segundo

GHOST – *General Haptic Open Software ToolKit*

HMD – *Head-Mounted Display*

J3D – *Java 3D*

JVM – *Java Virtual Machine*

KISMET – *Kinematic Simulation, Monitoring and Off-Line Programming Environment for Telerobotics*

MSJVM – *Microsoft Java Virtual Machine*

PC – *Personal Computer*

RV – Realidade Virtual

VESTA – *Virtual Environments for Surgical Training and Augmentation*

VRML – *Virtual Reality Modeling Language*

WTK – *WorldToolKit*

SUMÁRIO

1. INTRODUÇÃO

Uma dificuldade encontrada ainda hoje em relação ao desenvolvimento de aplicações de Realidade Virtual (RV) é a seleção adequada de tecnologia de *software* e *hardware* para o desenvolvimento de ambientes virtuais. Fatores como custo de desenvolvimento e custo computacional (desempenho) são itens importantes a serem considerados de acordo com o tipo de aplicação a ser desenvolvida.

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).

Na etapa da detecção de colisão, a simulação deve emitir uma resposta ao encontro entre objetos, como a deformação, um salto, restrição ao movimento ou mesmo produzir forças e vibrações (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. Além disso, em aplicações médicas, questões de precisão, tempo de resposta, conhecimento do local de impacto e verificação de interpenetração entre os objetos são fundamentais para a aproximação do procedimento real.

A deformação, por sua vez, permite aumentar o realismo em simulações, representando mudanças na forma dos objetos sempre que uma colisão ocorre. Assim como para a detecção de colisão, existem diversos estudos sobre métodos que analisam a deformação a partir da aplicação de uma força.

1.1 Objetivos

O objetivo final deste trabalho foi realizar uma comparação de tecnologias de *software* para construir aplicações de RV. Como é inviável realizar tal comparação envolvendo todas as variáveis que devem ser consideradas em uma aplicação de RV, o trabalho foi direcionado para a detecção de colisão, que consiste em um aspecto muito importante em qualquer aplicação. Ainda 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 projetos com a finalidade de identificar as tecnologias de *hardware* e *software* citadas na literatura, focando o desenvolvimento de ferramentas de RV para treinamento médico.

A partir do levantamento realizado, foram escolhidas duas tecnologias de *software* para análise: linguagem Java com a biblioteca Java 3D versão 1.3.1 (JAVA, 2003) e o WorldToolKit *Release* 8 (SENSE8, 2003), um *software* comercial para criação de aplicações tridimensionais (3D) e mundos virtuais. Para implementação dos métodos e levantamento dos dados para avaliação, foi selecionado o projeto descrito em Lima et al *(*2004) que apresenta um protótipo de ferramenta para simulação do exame de punção da mama.

1.2 Estrutura do trabalho

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

Capítulo 2 – Realidade Virtual e Aplicações em Medicina: são apresentados conceitos de Realidade Virtual, seu uso na medicina, incluindo exposição de vários projetos e uma análise tecnológica.

Capítulo 3 – Detecção de Colisão: são descritos os principais métodos para detecção de colisão, como hierarquia de volumes limites, detecção de colisão em objetos rígidos e deformáveis, utilização de métodos e algoritmos para detecção de colisão e deformação em aplicações médicas.

Capítulo 4 – Tecnologias de *Software* para o Desenvolvimento de Aplicações em Realidade Virtual: são introduzidas as bibliotecas e *softwares* mais comumente utilizados na implementação de aplicações baseadas em Realidade Virtual e selecionados para o desenvolvimento do projeto;

Capítulo 5 – Testes e Implementações: são discutidos os testes com os métodos oferecidos pelas tecnologias selecionadas para detecção de colisão e implementação de refinamentos.

Capítulo 6 – Resultados e Discussões: é realizada uma discussão acerca dos resultados obtidos nos testes e implementações realizadas por via de uma análise comparativa.

Capítulo 7 – Conclusões e Trabalhos Futuros: é apresentada a conclusão do trabalho e propostas para sua continuidade.

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

2. REALIDADE VIRTUAL E APLICAÇÕES EM MEDICINA

Há um vasto conjunto de aplicações de RV nas mais diversas áreas. Em muitos casos, essas aplicações têm revolucionado a forma de interação das pessoas com sistemas complexos, proporcionando melhor desempenho e reduzindo custos. A área médica, por sua vez, tem tido uma atenção especial por parte dos pesquisadores em RV. As aplicações voltadas para visualização científica, treinamento e simulação de procedimentos médicos têm sido o foco da maioria dos projetos.

Como explicado na introdução desta dissertação, pretende-se neste trabalho realizar uma análise comparativa entre duas tecnologias de software para a construção de ferramentas de RV. Para diminuir o escopo do trabalho, concentrou-se a atenção em aplicações para a área médica. Neste capítulo, então, serão apresentados os principais conceitos de RV, os aspectos da RV aplicada à medicina e uma variedade de projetos de simulação e treinamento médico. A finalidade do capítulo é permitir ao leitor uma familiarização com tais aplicações, além de tecer um panorama global das tecnologias utilizadas nesses sistemas. A fim de basear a seleção das tecnologias de *software* utilizadas nos testes e implementações, na seção final é exposta uma análise tecnológica dos projetos estudados.

2.1 Conceitos de Realidade Virtual

O termo Realidade Virtual (RV) possui diversas definições, tanto na área acadêmica quanto na área comercial. De forma objetiva pode ser definida como uma interface mais 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. Para suportar esse tipo de interação o usuário pode utilizar dispositivos não convencionais, como capacetes de visualização e controle, e luvas de dados chamadas *datagloves* (NETTO et al, 2002).

De acordo com Latta e Oberg (1994), a RV envolve a criação e experimentação de ambientes. Seu objetivo central é colocar o usuário num ambiente que não é vivenciado normalmente ou facilmente, estabelecendo relações entre ambos.

O termo mundo virtual descreve um mundo digital criado por meio de técnicas de computação gráfica. A partir do momento que é possível interagir e explorar esse mundo por meio de dispositivos de entrada e de saída, ele se transforma em um Ambiente Virtual (AV) ou Ambiente de Realidade Virtual (NETTO et al, 2002).

Ainda segundo Netto et al (2002), para diferenciar um AV de uma animação CAD (*Computer Aided Design*) ou multimídia, é necessário que o AV seja orientado ao usuário considerando três aspectos: imersão, interação e envolvimento. A seguir, são apresentadas as definições destes aspectos, conforme Netto et al (2002).

A imersão deve proporcionar ao usuário a sensação de presença dentro do mundo virtual. Do ponto de vista da visualização, a RV pode ser considerada imersiva ou não. A RV imersiva baseia-se no uso de capacetes ou cavernas (salas em que paredes, teto e chão são telas de projeção), enquanto a não imersiva utiliza monitores. Admite-se também certo grau de imersão tendo dispositivos baseados em outros sentidos, como a audição e o tato, levando também em consideração a força de reação para uma imersão completa.

A interação está ligada à influência das ações do usuário no comportamento dos objetos, o que permite cativá-lo em função das mudanças que ocorrem de acordo com os seus comandos.

O envolvimento, por sua vez, está associado ao grau de motivação que o mundo virtual proporciona a este usuário, podendo ser passivo, como ler, ou ativo, como participar de um jogo com mais de um jogador.

A RV também pressupõe renderização em tempo real, isto é, atualização das imagens sempre que a cena sofre qualquer modificação, e inclusão da descrição funcional dos objetos, estendendo a descrição geométrica e topológica do CAD (NETTO et al, 2002).

Um histórico de aplicações baseadas em RV pode ser visto em diversos artigos, como Comeau e Bryan (1961), Hand (1994), Jacobson (1994), Pimentel e Teixeira (1995). Recentemente, projetos baseados em RV estão sendo desenvolvidos nas mais diversas áreas do conhecimento, como automação, planejamento e manutenção, treinamento e simulação, e concepção e visualização de dados (NETTO et al, 2002).

Em termos de tecnologia aplicada à medicina, as aplicações médicas baseadas em RV tornaram essa área importante, comercial e clinicamente (MACHADO, 2003). As aplicações voltadas para visualização científica, treinamento e simulação de procedimentos médicos têm sido o foco da maioria dos projetos. As seções 2.2 e 2.3 abordam de maneira ampla a aplicação de RV na medicina.

2.2 Aspectos da Realidade Virtual na medicina

As possibilidades de aplicação de RV na medicina são imensas, desde os estudos anatômicos, passando pela educação e treinamento, ao planejamento e simulação de procedimentos cirúrgicos.

Em 1994, Richard M. Satava relatava que as aplicações médicas do século 21, baseadas na infra-estrutura da informação, incluiriam cirurgia por telepresença, simuladores cirúrgicos baseados em RV, informática médica e reabilitação (SATAVA, 1994). Estas aplicações possibilitariam mais confiança e melhor desempenho, sem os temores associados à realização de procedimentos de risco.

Os avanços da tecnologia da computação gráfica 3D, em particular, o surgimento de poderosas tecnologias de *hardware*, trouxeram uma onda de interesse às aplicações de RV imersiva que visam suprir necessidades educacionais e de treinamento rapidamente crescentes da moderna sociedade da informação (PONDER et al, 2003).

Atualmente, os projetos de aplicações de RV na medicina mais comumente encontrados estão direcionados à construção de ferramentas de simulação para procedimentos cirúrgicos.

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).

Adicionalmente, usando técnicas de interação baseadas em RV, os futuros e atuais cirurgiões podem adquirir habilidades básicas dos instrumentos que manipulam, aprender novos procedimentos e determinar níveis de competência antes de enfrentar um paciente real.

Segundo Tsai et al (2000), diversos simuladores cirúrgicos baseados em RV estão sendo desenvolvidos para fornecer informação detalhada a respeito dos tecidos, das ferramentas e das ações simuladas dos cirurgiões. Muitos têm enfatizado a técnica de cirurgia minimamente invasiva, por se tratar de uma técnica cirúrgica que está se tornando comum pela rápida recuperação dos pacientes, mas que exige um treinamento intensivo por parte dos cirurgiões.

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

Na próxima seção são apresentadas algumas aplicações desenvolvidas em centros de pesquisa.

2.3 Aplicações de Realidade Virtual em medicina

Uma variedade de projetos de simulação e treinamento médico, em desenvolvimento ou em fase de testes, pode ser encontrada na literatura e na Internet. A maioria oferece sistemas interativos, com estímulo visual e tátil, e em tempo-real. A seguir, são apresentados alguns desses trabalhos, obedecendo a uma ordem cronológica.

Um tema importante de pesquisa é o desenvolvimento de simuladores para treinamento no diagnóstico de câncer. Em Burdea et al (1999) foi apresentado um simulador baseado em RV para diagnóstico de câncer de próstata. O diagnóstico virtual pode ser realizado por meio de um anel que fornece retorno tátil ligado ao simulador que seleciona um

de quatro diferentes casos: próstata normal, próstata aumentada, tumor em estado inicial (Figura 2-1) e tumor em estado avançado (Figura 2-2). Um grande diferencial para o simulador descrito está no fato de permitir gravação e reprodução dos exames simulados para posterior apresentação.

Tendo como componente-chave de *hardware* o dispositivo háptico *Phantom* da *Sensable Technologies* (Figura 2-3), a implementação do *software* usou a biblioteca OpenGL (OPENGL, 2003) para renderização dos modelos 3D e a biblioteca háptica GHOST (*General Haptic Open Software ToolKit*), também da *Sensable Technologies,* para a modelagem física, ambas rodando numa estação *Silicon Graphics* de alto desempenho.

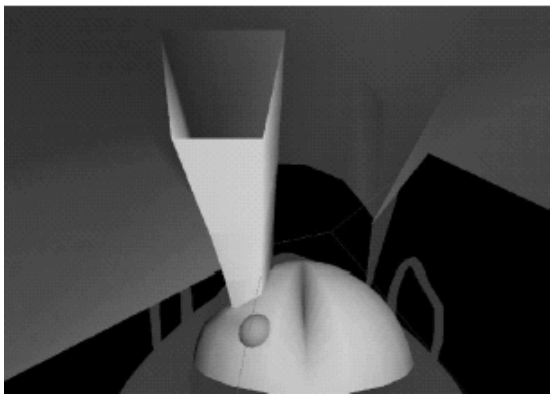


FIGURA 2-1: VISÃO INTERIOR DE UMA PRÓSTATA COM UM TUMOR EM ESTADO INICIAL (BURDEA ET AL, 1999).

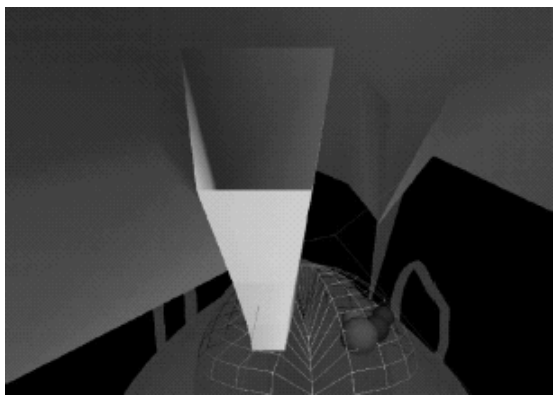


FIGURA 2-2: VISÃO INTERIOR DE UMA PRÓSTATA TRANSPARENTE COM UM TUMOR EM ESTADO AVANÇADO (BURDEA ET AL, 1999).

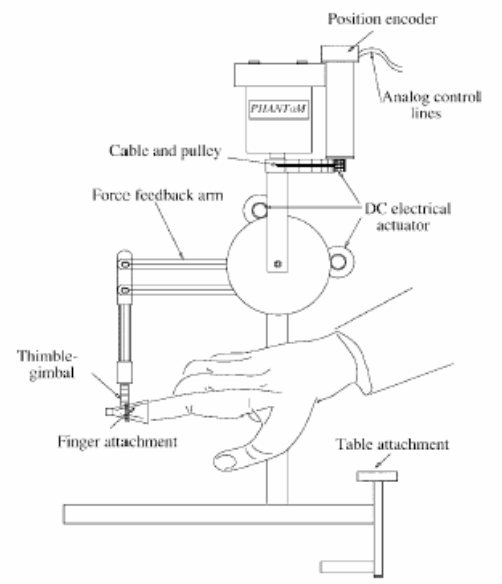


FIGURA 2-3: CONFIGURAÇÃO DO DISPOSITIVO HÁPTICO *PHANTOM* (BURDEA ET AL, 1999).

Gorman et al *(*2000) apresentaram um simulador para treinamento em acupuntura lombar. O objetivo foi evitar que erros de estudantes de medicina causassem mal a pacientes reais, como dores desnecessárias, tecidos danificados e ferimentos.

O simulador descrito consiste de uma agulha para acupuntura lombar passada pelo dorso de um manequim humano e unida a um dispositivo háptico *Phantom Desktop* da *Sensable Technologies* (Figura 2-4). O dispositivo háptico permite ao estudante sentir forças resistentes sobre uma trajetória correta ou incorreta como, por exemplo, numa colisão com osso.

A plataforma de desenvolvimento foi do tipo PC (*personal computer*) com dois processadores Pentium III 500 MHz e uma placa gráfica *Wildcat* 4000, mas de acordo com Gorman et al (2000), o sistema poderia ser executado adequadamente num Pentium III 450 MHz com uma placa gráfica moderada. Quanto à implementação, modelos 3D foram armazenados como arquivos VRML (*Virtual Reality Modeling Language*) (AMES et al, 1997) permitindo ilustrar como a agulha estava sendo inserida (Figura 2-5) e para os módulos

físico e gráfico foram utilizados, respectivamente, a biblioteca háptica GHOST e o *software* Worldtoolkit (SENSE8, 2003), além da utilização da biblioteca OpenGL.

Assim como o simulador para diagnóstico de câncer de próstata, o de treinamento em acupuntura lombar permite registrar todas as ações do usuário para futura análise do que aconteceu internamente durante o procedimento.

Simuladores para treinamento em laparoscopia (procedimento cirúrgico para exame da cavidade abdominal e pélvica) também são objetos de interesse e pesquisa, devido às imagens fornecidas por uma câmera no interior do paciente serem as únicas fontes de informação visual.

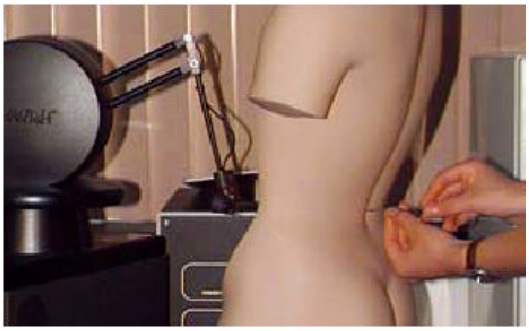


FIGURA 2-4: SIMULADOR PARA TREINAMENTO EM ACUPUNTURA LOMBAR (GORMAN ET AL, 2000).

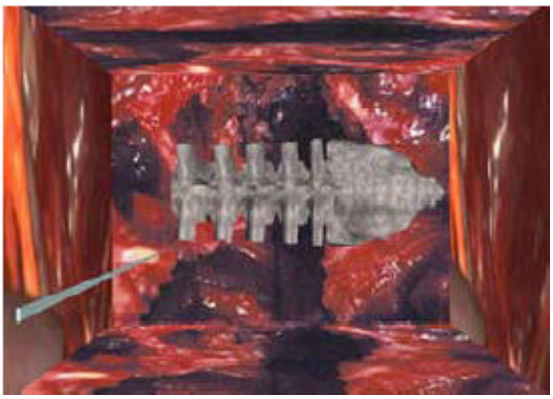


FIGURA 2-5: MODELO 3D DE UMA ESPINHA EM VRML (GORMAN ET AL, 2000).

Em Tendick et al *(*2000) foi descrito o projeto VESTA (*Virtual Environments for Surgical Training and Augmentation*), um AV desenvolvido pela Universidade da Califórnia para pesquisa no entendimento, avaliação e treinamento de habilidades em cirurgias laparoscópicas, incluindo percepção de habilidades motoras, espaciais e passos críticos de procedimentos cirúrgicos (Figura 2-6).

As implementações foram feitas na linguagem C com uso da biblioteca OpenGL, executando numa estação *Octane* da *Silicon Graphics* com 2 processadores R10000 de 250 MHz, incluindo uma interface háptica *Phantom* da *Sensable Technologies* de 3DOF modificada para 4DOF (Figura 2-7).

Outro tema para o desenvolvimento de simuladores é o treinamento de cirurgias ortopédicas. Em Sourina et al *(*2000) foi descrito um simulador chamado *Virtual Bone-setter*. Normalmente em treinamento de cirurgia ortopédica, estudantes fixam fraturas de ossos plásticos sintéticos (Figura 2-8) usando ferramentas cirúrgicas e implantes, mas ossos sintéticos de boa qualidade são caros e não há modelos disponíveis para todos os tipos de ossos. Por estes motivos, os cirurgiões do Departamento de Ortopedia do Hospital Geral de Singapura, em conjunto com a Escola de Ciência Aplicada da Universidade Tecnológica Nanyang, pensaram em usar o computador para treinamento de cirurgia ortopédica (Figura 2-9).

Implementado com o *software Renderware* da *Criterion* e projetado para executar em PC 150 MHz ou maior, 64 Mb de memória RAM e placa de vídeo 3D, o sistema apresentou um realismo aceitável sem a necessidade de dispositivos não convencionais. No entanto, permitiu obter melhor imersão a partir do uso de *mouse* 3D ou *pads* gráficos.

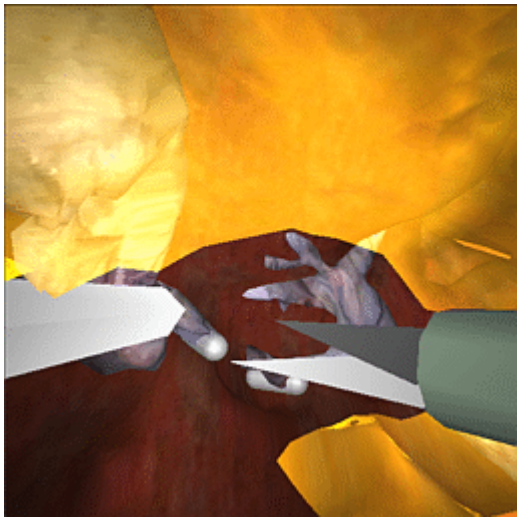


FIGURA 2-6: AMBIENTE SIMULADO PARA CIRURGIA LAPAROSCÓPICA (TENDICK ET AL, 2000).

FIGURA 2-7: INTERFACE HÁPTICA (TENDICK ET AL, 2000).

Ainda na área ortopédica, Tsai et al *(*2000) apresentaram um simulador capaz de reproduzir procedimentos cirúrgicos como corte, identificação, remoção e reposicionamento de estrutura, além de união de duas estruturas em uma e teste de colisão durante o movimento de uma estrutura (Figura 2-10).

O *software* foi desenvolvido em Visual C++ 5.0 para Windows em conjunto com a biblioteca OpenGL. Projetado para plataforma PC, na simulação utilizando um Pentium III 800 MHz com 256 Mb de memória RAM, permitiu o uso de óculos obturadores e um rastreador. Pela posição e altitude do rastreador, o sistema pode registrar dados espaciais para o instrumento virtual selecionado pelo usuário.

FIGURA 2-8: TÍPICO OSSO PLÁSTICO FRATURADO (SOURINA ET AL, 2000).

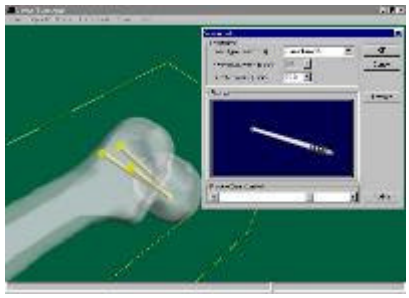


FIGURA 2-9: FIXAÇÃO DE UMA FRATURA VIRTUAL (SOURINA ET AL, 2000).

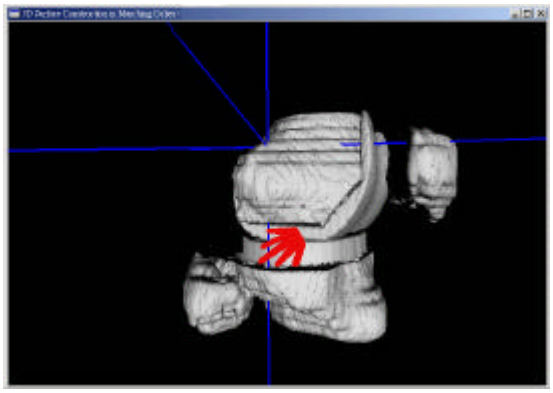


FIGURA 2-10: ARTOPLASTIA DE JOELHO PARA CORREÇÃO DE POSICIONAMENTO (TSAI ET AL, 2000).

A histeroscopia, procedimento onde o canal cervical e a cavidade uterina são estudados, também é tema de estudo para simuladores. Montgomery et al *(*2001) apresentaram um simulador cirúrgico para histeroscopia, desenvolvido pelo Centro Nacional de Biocomputação da Universidade Stanford da Califórnia, usado para análise e planejamento cirúrgico, treinamento de microcirurgia, simulação de suturas e outras aplicações. O sistema descrito permite ao usuário visualizar dados estereoscópicos ou monoscópicos como objetos

wireframe (visualização dos pontos e linhas que descrevem o modelo), sólidos ou semitransparentes (Figura 2-11).

Desenvolvido na linguagem C++ com a biblioteca OpenGL, a aplicação foi projetada para executar em diferentes plataformas computacionais. Na simulação foi utilizada uma estação gráfica Ultra60 Elite3D da Sun com 2 processadores UltraPARC 500 MHz e 1 Gb de memória RAM. O *software* é *multithreaded*, sendo um processador dedicado aos gráficos e outro à comunicação, detecção de colisão e simulação.

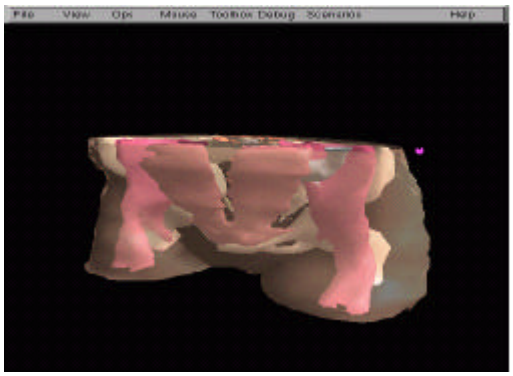


FIGURA 2-11: RENDERIZAÇÃO DE UM CONJUNTO DE DADOS (MONTGOMERY ET AL, 2001).

Outro exemplo de simulador baseado em RV é o *EyeSi* - simulador para cirurgia intraocular (WAGNER et al, 2002). O projeto foi baseado numa instalação mecânica completamente modelada para fornecer todas as percepções sensoriais do olho ao cirurgião em treinamento (Figura 2-12). O olho mecânico tem o mesmo grau de liberdade de rotação do olho humano, sendo que o efeito dos músculos foi modelado por um conjunto de molas sobre cada eixo de rotação. Além disso, a visualização realista do cenário de operação incluía efeitos de iluminação e sombras.

O EyeSi possui diversos módulos de treinamento que variam das tarefas abstratas a operações inteiras. Tarefas abstratas enfatizavam aspectos particulares do treinamento como, por exemplo, manipulação bimanual de instrumentos, distâncias estimadas por sombras ou

trabalho com um tipo especial de instrumento. Medidas de desempenho foram colhidas utilizando um PC Athlon 1,4 GHz AGP4x com uma placa gráfica NVIDIA *GeForce* 2 GTS.

FIGURA 2-12: CIRURGIA VIRTUAL NO OLHO MECÂNICO (WAGNER ET AL, 2002).

Há também o simulador descrito em Webster et al *(*2003) (Figura 2-13), que foi desenvolvido utilizando a estação para simulação de cirurgia laparoscópica da *Immersion Medical* (Figura 2-14), a qual permitiu um alto nível de retorno háptico.

O *software* de treinamento executa sobre uma estação Windows XP com dois processadores Pentium 2.2 GHz e uma placa gráfica NVIDIA *GeForce*. A aplicação realiza chamadas em rotinas da biblioteca OpenGL para renderizar ferramentas gráficas 3D.

Ainda devem ser citadas ferramentas que dão suporte ao desenvolvimento de simuladores como, por exemplo, o KISMET (*Kinematic Simulation, Monitoring and Off-Line Programming Environment for Telerobotics*) (KISMET, 2003) e o sistema LapSim (LAPSIM SYSTEM, 2003), ambos comerciais.

Projetado para planejamento, simulação, programação e monitoramento de tarefas teleoperacionais, tendo aplicação em simulação médica (Figura 2-15) e visualização científica, o KISMET permite visões de diversas cenas ao mesmo tempo com níveis interativamente selecionáveis de detalhes. Implementado na linguagem C com a biblioteca OpenGL, o *software* está disponível para estações *Silicon Graphics*, mas com versão beta para PC com Sistema Operacional Windows NT 4. Os dispositivos hápticos aceitos são:

Laparoscopic Impulse Engine da *Immersion Medical*, *Phantom* da *Sensable Technologies* e o *HIT Force Feedback Device* (KISMET, 2003).

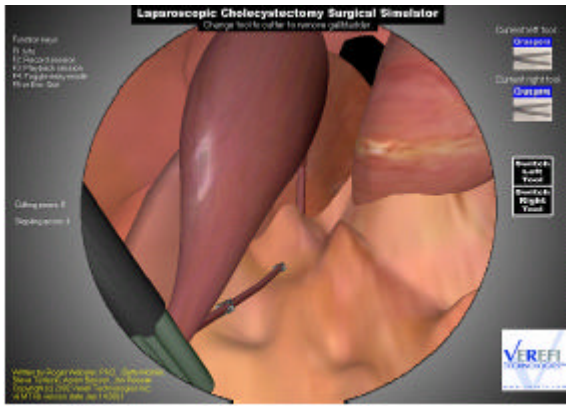


FIGURA 2-13: SIMULAÇÃO DE CIRURGIA LAPAROSCÓPICA (WEBSTER ET AL, 2003).

FIGURA 2-14: ESTAÇÃO PARA SIMULAÇÃO DE CIRURGIA LAPAROSCÓPICA (IMMERSION MEDICAL, 2003).

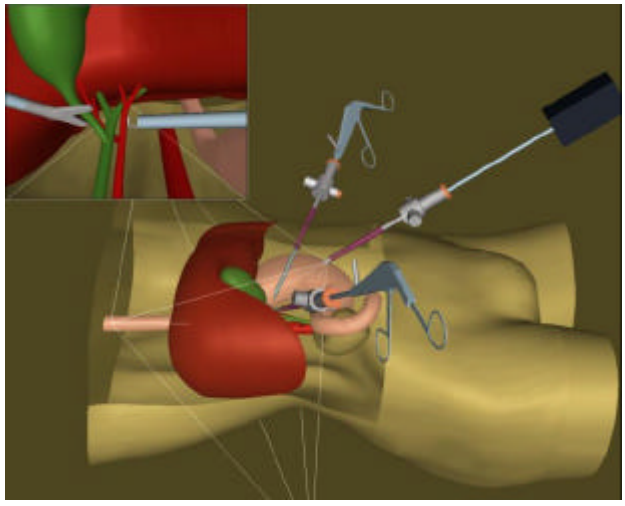


FIGURA 2-15: SIMULAÇÃO MÉDICA COM SUPORTE DO KISMET (KISMET, 2003).

O sistema LapSim, projetado para treinamento e simulação de cirurgias laparoscópicas, é dividido em três módulos, sendo um para habilidades básicas como corte e sutura (Figura 2-16), um para o procedimento de dissecação (Figura 2-17) e outro, mais recente, para cirurgias ginecológicas (Figura 2-18).

Disponível tanto para sistema monousuário quanto para sistema multiusuário, LapSim requer um PC com um processador Pentium III 1GHz ou com dois processadores Pentium III 500 MHz, com 128 Mb de memória RAM, 20 Gb de HD e placa gráfica NVIDIA *GeForce* 1, 2 ou 3, com Sistema Operacional Windows 2000 ou XP. Os dispositivos hápticos aceitos são: Interface Laparoscópica Virtual (Figura 2-19) e/ou Estação Cirúrgica Laparoscópica, ambos da *Immersion Medical*.

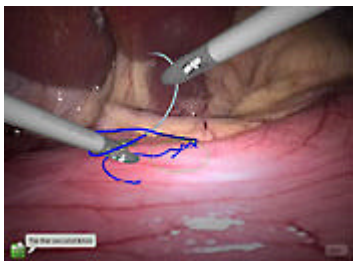


FIGURA 2-16: SUTURA NO MÓDULO BASICSKILLS (LAPSIM SYSTEM, 2003).

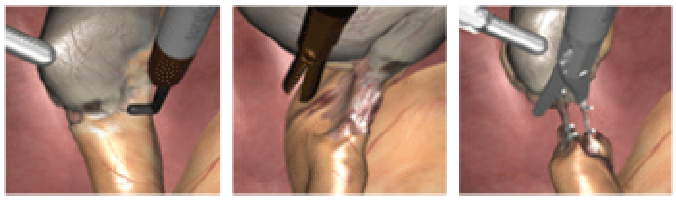


FIGURA 2-17: DISSECAÇÃO NO MÓDULO DISSECTION (LAPSIM SYSTEM, 2003).

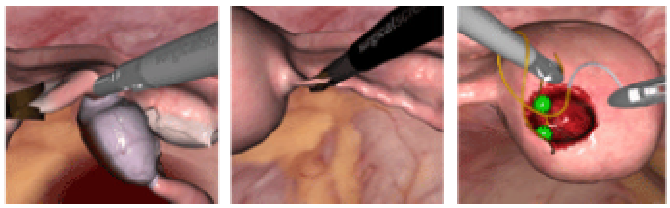


FIGURA 2-18: CIRURGIA GINECOLÓGICA NO MÓDULO GYN (LAPSIM SYSTEM, 2003).

FIGURA 2-19: INTERFACE LAPAROSCÓPICA VIRTUAL (IMMERSION MEDICAL, 2003).

2.3.1 Projetos no Brasil

No Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo, com colaboração do Departamento de Pediatria do Instituto da Criança, foi desenvolvido um simulador de coleta de medula óssea para treinamento pediátrico (Figura 2-20). Implementado na linguagem C++, utilizando as bibliotecas OpenGL e GHOST, o sistema pode ser executado num PC com Sistema Operacional Windows NT, equipado com um mínimo de 256 Mb de memória RAM, óculos 3D e dispositivo háptico *Phantom Desktop* (MACHADO, 2003).

Um dos motivos do projeto, além da não utilização de animais como cobaias, foi permitir maior usabilidade e disponibilidade no treinamento (MACHADO, 2003).

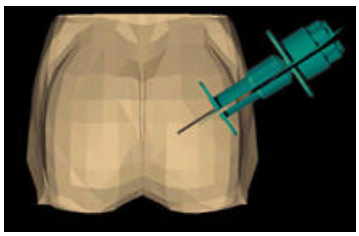


FIGURA 2-20: "AGULHA VIRTUAL" DO SIMULADOR DE COLETA DE MEDULA ÓSSEA (MACHADO, 2003).

Outro projeto nacional é o Vpat, um *framework* orientado a objetos para construção de pacientes virtuais, desenvolvimento de sistemas de visualização e exploração de dados médicos, em desenvolvimento pela Universidade Federal do Rio Grande do Sul (FREITAS et al, 2003).

Constituído de classes básicas como pontos, vértices, vetor, matriz e cor, o VPat possui classes específicas para visualização volumétrica e modelagem anatômica de articulações, cujas funcionalidades podem ser compartilhadas e estendidas. As classes foram implementadas na linguagem C++ independente da plataforma, sendo as de visualização volumétrica implementadas sobre a biblioteca FLTK (*Fast Light ToolKit*) e as para modelagem anatômica de articulações na *Open Inventor*.

Além da geração de modelos de representação de seres humanos virtuais para uso em aplicações de computação gráfica na área médica, o projeto visa permitir o melhor entendimento da forma humana, suas funções e seu desenvolvimento, com objetivo de desenvolver simuladores que permitam a interação com corpos virtuais, pelo uso de equipamentos de RV (FREITAS et al, 2003).

Como pode ser percebido, o desenvolvimento desta área de pesquisa no Brasil ainda é incipiente. Por isso, a importância de um estudo mais aprofundado de tecnologias de *software* e *hardware* disponíveis para implementação de aplicações nesta área.

2.4 Análise tecnológica

Como pôde ser observado na seção anterior, a simulação cirúrgica baseada em RV deve permitir ao usuário a percepção de situações que mais lhe aproximam do real. Neste

sentido, cada projeto de simulador envolve uma série de requisitos computacionais que sustentam pontos específicos da sua implementação e contribuem para o realismo do AV.

De acordo com Machado (2003), estes pontos específicos podem ser: detecção de colisão entre objetos, deformação dos modelos, aspectos de visualização, desempenho computacional, simulação de forças com retorno tátil, custo do sistema, reconstrução volumétrica a partir de dados reais, avaliação objetiva do procedimento, avaliação do usuário e ergonomia.

Muitos simuladores são projetados para executar em plataformas do tipo PC com Sistema Operacional Windows, pois este é um importante requisito para se ter um menor custo, além da fácil assimilação e manuseio por parte do usuário (MACHADO, 2003; MCCARTHY e HOLLANDS, 1997). Mas dependendo do nível de detalhamento dos objetos e da qualidade das imagens, pode haver necessidade de plataformas computacionais de processamento paralelo ou de alto desempenho gráfico, como estações *Silicon Graphics*, que passam a ter um custo mais elevado.

Simuladores que reúnem estereoscopia, deformação e interação háptica, geralmente requerem plataformas de alto desempenho gráfico, mas montados em plataformas do tipo PC utilizam modelos simplificados, nenhuma deformação ou deformação otimizada e retorno háptico ativo ou passivo (quando sons indicam o contato) (MACHADO, 2003).

A Tabela 2-1 apresenta um resumo dos recursos de *hardware* e *software* utilizados nas ferramentas pesquisadas. Como pode ser notado, a maioria dos simuladores são implementados na linguagem C com uso das bibliotecas OpenGL e GHOST, além de serem projetados para plataformas do tipo PC.

Aspectos como qualidade de renderização e renderização sem *hardware* gráfico especial foram motivos para a escolha do ambiente de execução e programação nos projetos descritos em Tsai et al (2000) e Sourina et al (2000), respectivamente. Em Montgomery et al

(2001), o uso da linguagem C++ com a biblioteca OpenGL foi justificado por permitir a execução em diversas plataformas computacionais. Já em Machado (2003), a escolha baseou-se na compatibilidade com a plataforma de execução adotada e no tipo do usuário final, considerando também a viabilidade do custo de implantação e da necessidade de um ambiente de fácil assimilação e manuseio.

Apesar de outros autores citados não apresentarem justificativas para o ambiente escolhido, a Tabela 2-1 comprova a busca por menores custos de desenvolvimento e execução a partir da utilização de plataformas de baixo custo (PCs) e ambientes de programação e execução padrões de mercado, como a linguagem C, a biblioteca OpenGL e o Sistema Operacional Windows. Entretanto, o uso de dispositivos não convencionais como óculos, rastreadores e dispositivos hápticos, bem como a utilização de softwares comerciais, pode manter alto o custo final do projeto.

Tabela 2-1: Simuladores: plataformas para execução/simulação, ambientes de programação e execução.

Simulador/Ferramenta para Simulação	**Plataforma de Execução/Simulação**	**Programação e Execução**
VR-Based Training for The Diagnosis of Prostate Cancer (BURDEA et al, 1999)	Estação Silicon Graphics; Dispositivo háptico: Phantom (Sensable Technologies).	GHOST e OpenGL
A VE Testbed for Training Laparoscopic Surgical Skills (TENDICK et al, 2000)	Estação Silicon Graphics Octane; DP 250 MHz R10000.	C; OpenGL
An Orthopedic VR Surgical Simulator (TSAI et al, 2000)	PC Pentium III 800 MHz; 256 Mb RAM;	VISUAL C++ 5.0; OpenGL
	Óculos obturadores e um rastreador.	Windows
A Prototype Haptic Lumbar Puncture Simulator (GORMAN et al, 2000)	PC; DP PIII 500 MHz; Placa gráfica Wildcat 4000 OpenGL accelerator;	WorldToolkit;
	Dispositivo háptico: Phantom Desktop (Sensable Technologies).	GHOST e OpenGL; VRML
Virtual Bone-setter (SOURINA et al, 2000)	PC 150 MHz ou melhor; 64 Mb RAM; Placa de vídeo 3D.	Criterion Renderware
Surgical Simulator for Hysteroscopy (MONTGOMERY et al, 2001)	Estação Sun Ultra60 Elite 3D Graphics; DP UltraPARC 500 MHz; 1GB RAM.	C++; OpenGL
EyeSi - VR-Simulator for Eye Surgery (WAGNER et al, 2002)	PC Athlon 1,4 GHz AGP4x; Placa gráfica NVidia GeForce 2 GTS.	
Simulador para Coleta de Medula Óssea (MACHADO, 2003)	PC; 256 Mb RAM;	C++; GHOST e OpenGL
	Óculos 3D; Dispositivo háptico: Phantom Desktop (Sensable Technologies).	Windows NT
A Haptic Surgical Simulator for	PC; DP Pentium 2.2 GHz; Placa gráfica Nvidia GeForce;	OpenGL; Windows XP
Laparoscopic Cholecystectomy (WEBSTER et al, 2003)	Dispositivo háptico: Estação Cirúrgica Laparoscópica (Immersion Medical)	
KISMET (KISMET, 2003)	Estação Silicon Graphics e PC (em teste);	C; OpenGL
	Dispositivos hápticos: Laparoscopic IMPULSE ENGINE (Immersion Medical),	IRIX 5.3; Windows NT 4.0
	Phantom (Sensable Technologies) e HIT Force Feedback Device.	
LapSim System (LAPSIM SYSTEM, 2003)	PC Pentium III 1 GHz; 128 Mb RAM; 20 Gb de HD;	Windows 2000 ou XP
	Placa gráfica Nvidia GeForce 1,2 ou 3.	
	Dispositivos hápticos aceitáveis: Interface Laparoscópica Virtual e/ou	
	Estação Cirúrgica Laparoscópica (Immersion Medical)	
VPat - Framework para Construção de Pacientes Virtuais (FREITAS et al, 2003)	Plataforma independente	C++; FLTK e Open Inventor

3. DETECÇÃO DE COLISÃO

Sendo o problema da detecção de colisão importante para o realismo de um AV, nesta etapa é necessário que a simulação emita uma resposta ao encontro entre objetos. Além disso, em simulação de procedimentos médicos, é importante saber onde ocorreu a colisão e se houve, ou não, interpenetração entre objetos (MACHADO e ZUFFO, 2003).

Neste capítulo são abordados métodos e algoritmos comumente utilizados na detecção de colisão e encontrados freqüentemente na literatura e na Internet, além de um levantamento das soluções adotadas pelos projetos descritos anteriormente.

3.1 Abordagem hierárquica (uso de volumes limites)

A abordagem hierárquica, ou uso de volumes limites, é freqüentemente empregada para uma rápida detecção de colisão. Um volume limite representa a extensão espacial máxima de um objeto, podendo ser representada por um cubo ou esfera (SENSE8, 2003). O uso deste tipo de abordagem consiste em criar aproximações geométricas para objetos complexos, permitindo eliminar áreas dos objetos que não estão em contato. A seguir, são apresentadas algumas técnicas comumente encontradas na literatura, descritas em O'Sullivan et al (2001):

- *Octrees*: hierarquias criadas recursivamente subdividindo o volume que contém um objeto em oito octantes (cubos ou esferas) e retendo somente aqueles octantes que contêm alguma parte do objeto original (Figura 3-1). Dependendo da quantidade de dados a serem armazenados, esta técnica pode requerer alto esforço computacional.

- *Sphere-trees*: as principais vantagens do uso de esferas são que elas possuem rotação invariável, além do cálculo da distância e sobreposição entre elas ser mais simples. A desvantagem é que esferas não aproximam certos tipos de objetos eficientemente, fato que pode comprometer a precisão do método para detecção de colisão (Figura 3-2).
- *OBB-trees*: estas hierarquias consistem de volumes limites orientados com o objeto (*Oriented Bounding Boxes*). A vantagem está na proximidade com a forma do objeto. Apesar de ter bom desempenho em detecções de colisões complexas, são mais difíceis de criar e manipular (Figura 3-3).
- *AABB-trees*: constituem de volumes limites alinhados aos eixos (*Axis Aligned Bounding Boxes*). Estas hierarquias são mais simples de criar e realizar testes de sobreposição. Entretanto, desvantagens ocorrem quando se tem "objetos na diagonal", isto é, não alinhados aos eixos (Figura 3-4).

Existem outras abordagens hierárquicas, como K-DOPs e ShellTrees, mas estas aproximações são consideradas generalizações ou variações das anteriormente citadas (O'SULLIVAN et al, 2001).

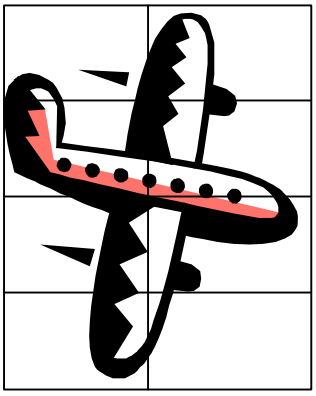


FIGURA 3-1: EXEMPLO DA ABORDAGEM HIERÁRQUICA OCTREE.

FIGURA 3-2: EXEMPLO DA ABORDAGEM HIERÁRQUICA SPHERE-TREE.

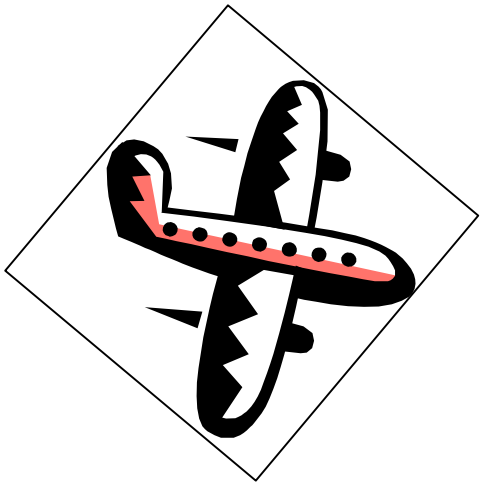


FIGURA 3-3: EXEMPLO DA ABORDAGEM HIERÁRQUICA OBB-TREE.

FIGURA 3-4: EXEMPLO DA ABORDAGEM HIERÁRQUICA AABB-TREE.

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, também conhecidos como superfícies flexíveis. 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.

Na detecção de colisão em objetos rígidos, podem ser utilizadas três aproximações:

- Subdivisão hierárquica do espaço: o ambiente é dividido em um espaço hierárquico e os objetos são aglomerados hierarquicamente de acordo com as regiões que se localizam. Quando um objeto presente no ambiente altera a sua posição, movendo-se para outra região, somente os objetos da nova região precisam ser checados para colisão com o objeto penetrante.
- Subdivisão hierárquica do objeto: cada objeto é subdividido em uma hierarquia de volumes limites. Neste caso, a colisão é determinada se a hierarquia de volumes limites de um objeto intersecta com a de outro.
- Cálculo incremental da distância: a distância mínima ou características mais próximas (lados, bordas e vértices) entre cada par de objetos é incrementada seguidamente de modo que suas colisões possam ser determinadas eficientemente.

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.

Em Nedel (1992), por exemplo, uma superfície deformável é simulada por um modelo geométrico que consiste de uma malha retangular representada por uma matriz de pontos x, y, z. A colisão de uma superfície deformável com objetos rígidos é calculada por uma função que verifica se cada ponto da malha está dentro, fora ou na superfície deste. Quando dentro do objeto rígido, é aplicada uma função que leva o ponto para a superfície do mesmo.

Em geral, se a deformação de um objeto for baseada em atualizar um modelo de polígonos, os métodos de detecção de colisão correspondentes podem ser mais eficientes porque envolvem somente a atualização da hierarquia limitada enquanto o objeto se deforma.

3.3 Utilização de métodos e algoritmos para detecção de colisão

Vários projetos encontrados na literatura buscam uma detecção de colisão precisa e com o menor custo computacional possível a partir de refinamentos ou combinação de métodos e algoritmos já existentes.

Em Garcia-Alonso et al (1994), por exemplo, foram utilizados três métodos para determinar uma colisão entre dois objetos:

- Minimax – os objetos são envolvidos de acordo com suas coordenadas locais (*container* local). Em seguida, o *container* local é envolvido por um outro, agora alinhado aos eixos globais do sistema (*container* minimax). A possível colisão é calculada comparando-se as coordenadas mínimas e máximas de cada objeto em cada eixo.

- Voxels – cada objeto é envolvido em um paralelepípedo e este é dividido em pequenos cubos idênticos. Os voxels vazios são desconsiderados, calculando a possível colisão em relação à interferência entre os voxels restantes dos objetos.
- Faces – cada objeto é representado pelas figuras geométricas que compõem suas faces. A colisão é detectada se algum par de facetas (um de cada objeto) se intercepta.

Ponamgi et al (1995) combinaram um algoritmo baseado no uso de volumes limites com aproximações incrementais de suas características (lados, bordas e vértices) para detecção de colisão entre sólidos em ambientes dinâmicos (Figura 3-5).

No projeto V-COLLIDE, descrito em Hudson et al (1997), houve uma combinação de recursos pertencentes às 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 estágio, algoritmos presentes na I-COLLIDE são usados para filtrar colisões entre um grande número de objetos, determinando pares de objetos que estejam potencialmente colidindo. Numa segunda fase, algoritmos da biblioteca RAPID são aplicados para determinar se os objetos recebidos do primeiro estágio estão realmente colidindo.

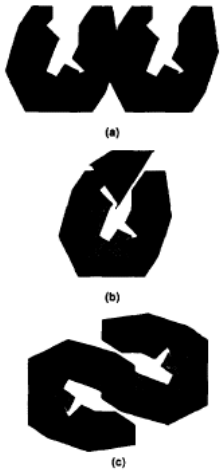


FIGURA 3-5: COLISÕES ENVOLVENDO DIFERENTES CARACTERÍSTICAS (PONAMGI ET AL, 1995).

3.4 Detecção de colisão e deformação em aplicações médicas

Como citado anteriormente, os dispositivos hápticos procuram estimular a sensação tátil e/ou de força, requerendo uma sofisticada interação eletromecânica com o corpo do usuário, envolvendo a compreensão e simulação de forças apropriadas. Estes requisitos exigem programas específicos para monitoramento das informações de interação, envolvendo rotinas de controle e computação gráfica.

Detecção de colisão e deformação são os itens mais complexos e dependentes das informações de interação monitoradas, estando presentes na modelagem física. De acordo com Machado (2003), modelagem física é a fase responsável pela determinação do comportamento dinâmico dos objetos do mundo virtual e controle do dispositivo háptico.

Em simulações de procedimentos cirúrgicos, instrumentos têm que interagir com modelos de organismos deformáveis com realismo. Este requisito exige métodos exatos para detecção de colisão entre ferramentas e organismos (TENDICK et al, 2000).

Burdea et al (1999), por exemplo, utilizaram o cálculo de distância entre vértices para a detecção de colisão entre a superfície de uma próstata virtual e um dedo virtual. A magnitude da deformação resultante era determinada pela distância do dedo em relação à superfície, o qual dava a ilusão que a próstata era graficamente comprimida pelo dedo.

Em Tendick et al (2000), é apresentada uma ferramenta para avaliação e treinamento de habilidades em cirurgias laparoscópicas, sendo que o espaço de trabalho foi subdividido em uma grade de voxels de mesmo tamanho. Num pré-processamento, esses voxels armazenavam os vértices de todas as ligas deformáveis neles contidas. No processo de detecção de colisão eram definidas áreas dos modelos de ferramentas cirúrgicas que poderiam vir a ter contato com o tecido. A cada passo da simulação era verificado se havia vértices na mesma célula da área selecionada e, em caso positivo, a distância era calculada para ser

comparada a um limiar que determinava a existência ou inexistência de colisão entre os objetos. O mapeamento de objetos em voxels também foi o método utilizado por TSAI et al (2000) para detectar a colisão entre ossos, próteses, vasos e nervos.

Já Wagner et al (2002) utilizaram uma aproximação baseada em imagem, uma forma de particionamento do espaço realizada no processo de rasterização de um sistema gráfico 3D. De acordo com o estudo realizado, a aproximação geométrica para detecção de colisão mostrou-se imprópria para a interação com objetos deformáveis, principalmente no critério desempenho.

A abordagem hierárquica com utilização de esferas foi o método utilizado por Montgomery et al (2001) no simulador cirúrgico para histeroscopia. Neste projeto, o algoritmo adotado sofreu algumas modificações para suportar objetos deformáveis e aumentar o desempenho no retorno háptico.

Em Machado e Zuffo (2003), a opção adotada foi o refinamento das rotinas de detecção de colisão contidas na biblioteca háptica GHOST. As rotinas de detecção de colisão foram modificadas para a obtenção da sensação de rigidez ao penetrar a agulha virtual nos modelos dos tecidos, estabelecendo diferentes graus de resistência para cada camada.

Outra etapa importante na modelagem física é a deformação dos objetos virtuais. A deformação oferece maior realismo à simulação, pois permite representar alterações na forma dos objetos na ocorrência de alguma colisão.

Conforme Machado (2003), as deformações dos objetos virtuais podem ser geométricas ou baseadas na física. As deformações geométricas são baseadas em manipulações de dados geométricos como vértices e pontos, já as baseadas na física envolvem o comportamento dos objetos sob efeito de alguma força física, seja ela interna ou externa.

Diversos trabalhos, como o descrito em Dimaio e Salcudean *(*2002), estudam métodos que quantifiquem a força aplicada e a deformação resultante em diversos meios. No

trabalho citado, o estudo tem como base a inserção de uma agulha virtual em vários tecidos e sua trajetória em múltiplas dimensões (Figura 3-6 e 3-7).

FIGURA 3-6: INSERÇÃO DE AGULHA EM TECIDOS MACIOS (DIMAIO E SALCUDEAN, 2002).

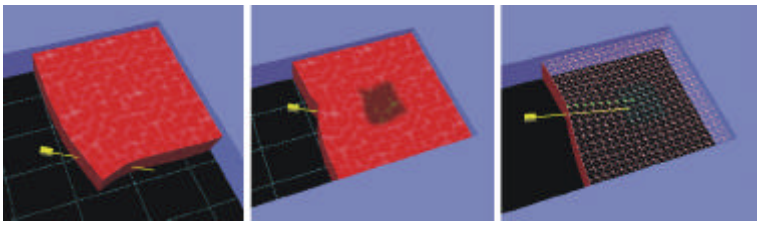


FIGURA 3-7: INSERÇÃO DE AGULHA NUM AMBIENTE PLANO (DIMAIO E SALCUDEAN, 2002).

Apesar de serem fatores importantes na simulação de procedimento médicos, resultados relacionados à precisão e custo computacional dos métodos utilizados para detecção de colisão e deformação, não foram discutidos pelos autores nos trabalhos citados. Por este motivo, a Tabela 2-1 não apresenta estes itens. O capítulo 5, por sua vez, apresenta uma análise baseada nestas questões, discutindo acerca dos métodos existentes nas tecnologias de *software* adotadas e o desenvolvimento de refinamentos.

4. TECNOLOGIAS DE *SOFTWARE* PARA O DESENVOLVIMENTO DE APLICAÇÕES EM REALIDADE VIRTUAL

Para a criação de qualquer AV, seja ele voltado para aplicação médica ou não, é necessário verificar os componentes de *software* que permitirão a implementação das funcionalidades requeridas da aplicação.

A escolha, normalmente, fica entre uma linguagem de programação (C, Java, etc.) em conjunto com uma biblioteca que forneça métodos de apresentação, interação e modificações no AV como, por exemplo, Java 3D (J3D), OpenGL ou WorldToolKit (WTK). Características das bibliotecas OpenGL e Java 3D, citadas e utilizadas no projeto, assim como o WTK, são detalhadas a seguir.

4.1 OpenGL

OpenGL (*Open Graphics Library*) é uma biblioteca que possui um conjunto completo de funções de baixo nível para desenvolvimento de aplicações interativas 2D e 3D, sendo indicada para os seguintes segmentos de mercado (OPENGL, 2003):

- CAD.
- Entretenimento.
- Visualização científica.
- Produção e visualização de gráficos 2D e 3D em aplicações de RV.

OpenGL, por ser portável, não possui nenhuma rotina para gerenciamento de janelas, interação com o usuário, acessos de entrada e saída (não recebe teclado, *mouse*, *joysticks*).

Essas especificações são feitas pela linguagem de programação utilizada e dependerá também da plataforma. Dependendo do nível do aplicativo a ser criado será necessário o uso de bibliotecas diversas com fins específicos (BICHO et al, 2002).

Além do desenho de primitivas gráficas, tais como linhas e polígonos, OpenGL dá suporte à iluminação e sombreamento, mapeamento de textura, transparência, animação e muitos outros efeitos. Não existe um formato de arquivo OpenGL para modelos ou ambientes virtuais. OpenGL fornece apenas um pequeno conjunto de primitivas gráficas para construção de modelos: pontos, linhas e polígonos. Já a biblioteca GLU (que faz parte da implementação OpenGL) possui várias funções para modelagem, tais como superfícies quádricas e curvas (BICHO et al, 2002).

Quanto ao seu funcionamento (Figura 4-1), quando uma aplicação faz chamadas às funções OpenGL, os comandos são colocados em um *buffer* de comandos. Este *buffer* é preenchido com comandos, vértices, dados de textura, etc. Quando este *buffer* é "esvaziado", os comandos e dados são passados para o próximo estágio.

Após a etapa de aplicação das transformações geométricas e da iluminação é feita a rasterização, isto é, é gerada a imagem a partir dos dados geométricos de cor e textura. A imagem final é colocada no *frame buffer*, que é a memória do dispositivo gráfico (MANSSOUR, 2003).

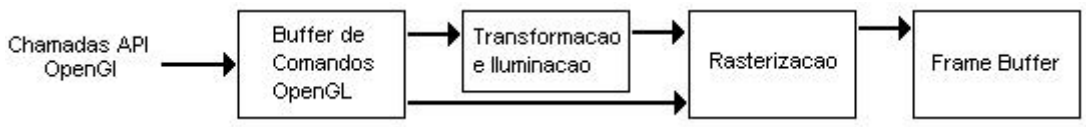


FIGURA 4-1: PIPELINE SIMPLIFICADO DA OPENGL (MANSSOUR, 2003).

Diante de suas funcionalidades, OpenGL tem se tornado um padrão amplamente adotado na indústria de desenvolvimento de aplicações. Este fato tem sido encorajado também pela facilidade de aprendizado, pela estabilidade das rotinas, pela boa documentação

disponível e pelos resultados visuais consistentes para qualquer sistema de exibição concordante com este padrão (OPENGL, 2003). Mas o fato de não ter suporte avançado para RV, não permitir integrar gráficos 3D com som 3D e deixar o desenvolvedor preso a detalhes de implementação (renderização), pode limitar seu uso.

4.2 Java 3D

J3D é uma biblioteca gráfica 3D baseada em grafos de cena e que utiliza todas as classes da linguagem Java, permitindo ao desenvolvedor se dedicar mais à criação (composição de cenas) do que aos problemas de baixo nível (detalhes de renderização), sendo indicada para os seguintes segmentos de mercado (JAVA, 2003):

- Criação de conteúdo e desenvolvimento de páginas *web* 3D.
- Produção e visualização de gráficos 2D e 3D em aplicações RV.
- Entretenimento.
- Gráficos de negócios, *data mining* e apresentações.
- Visualização Científica.
- Animação.

J3D possui finalidades importantes como: fornecer um bom conjunto de características para a criação de mundos 3D, estabelecer um paradigma de programação orientada a objeto de alto nível e permitir acomodar uma grande variedade de formatos de arquivo, tais como formatos CAD, formatos de intercâmbio, VRML 1.0 e VRML 2.0.

Em J3D os elementos gráficos são construídos como objetos que podem ser tratados tanto individualmente quanto em grupo (BICHO et al, 2002). Em função disto, os objetos são interligados na forma de árvore, numa estrutura denominada *grafo de cena* (Figura 4-2).

Um grafo de cena é uma estrutura de dados que descreve uma cena tridimensional na forma de nós ligados. 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. Para simplificação, pode-se pensar nesta estrutura como uma árvore, com os nós organizados de forma hierárquica possibilitando referências cíclicas.

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 e integração com a Internet, além da gratuidade e portabilidade da linguagem Java (JAVA, 2003). Mas a falta de suporte para processamento de imagens, instalação separada do núcleo da linguagem Java e alto consumo de memória dependendo da aplicação pode não contribuir para o desenvolvimento de uma aplicação.

Atualmente, toda a instrução da J3D para o *hardware* passa pela OpenGL ou DirectX (JAVA, 2003). 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.

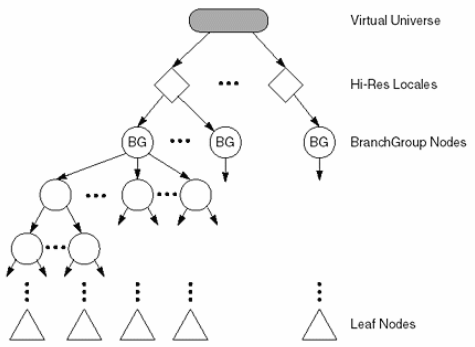


FIGURA 4-2: ESTRUTURA HIERÁRQUICA DO GRAFO DE CENA EM J3D (JAVA, 2003).

4.3 WorldToolKit

O WTK é um *software* comercial multi-plataforma (PCs de alto desempenho, SUN, DEC, HP e estações Silicon Graphics), portátil, de desenvolvimento de aplicações integradas 3D de alto desempenho e tempo-real, para fins científicos e comerciais (SENSE8, 2003). O WTK suporta simulações distribuídas em rede e um grande conjunto de dispositivos de interface, tais como HMDs (*Head-Mounted Displays)*, rastreadores e controladores de navegação, além de possibilitar ao desenvolvedor escrever seus próprios drivers de controladores.

Basicamente, o mundo virtual é criado usando a linguagem C e as bibliotecas fornecidas pelo WTK, podendo ser controlado com uma variedade de sensores de entrada, desde um simples mouse até um sensor com 6DOF. O WTK gerencia as tarefas da renderização, leitura de dispositivos de entrada, importação de objetos, e uma grande quantidade de funções de simulação, fazendo o trabalho de baixo nível, permitindo ao desenvolvedor ficar livre para se concentrar nos detalhes da aplicação como, por exemplo, composição de cenas. Projetado para aplicações de tempo-real, no núcleo da aplicação WTK encontra-se o loop da simulação que lê os sensores de entrada, atualiza os objetos e renderiza um novo frame da simulação.

Como na J3D, o desenvolvedor pode criar sua aplicação sem considerar detalhes dos recursos de hardware que deseja utilizar. Assim, pelo alto nível de abstração, a biblioteca J3D e o software WTK foram as tecnologias de *software* mais adequadas para análise dos métodos oferecidos para detecção de colisão e implementação dos refinamentos necessários, apesar da grande aplicabilidade da biblioteca OpenGL. Os capítulos seguintes apresentam todas as etapas realizadas para atingir o objetivo principal deste trabalho, assim como os resultados obtidos com as implementações.

5. TESTES E IMPLEMENTAÇÕES

Como mencionado anteriormente, após os estudos realizados sobre os vários projetos de RV para treinamento médico e o levantamento das abordagens mais comuns para detecção de colisão entre objetos, foi possível selecionar as tecnologias de *software* e *hardware* necessárias para realizar o estudo comparativo de soluções para detecção de colisão, conforme o escopo do trabalho.

O objetivo do trabalho aqui apresentado é tecer um estudo comparativo entre o comportamento de tecnologia de *software* no que diz respeito à detecção de colisão, preocupando-se especificamente com as aplicações para treinamento médico que exigem precisão e rapidez. Assim, foi escolhido um estudo de caso de treinamento médico para direcionar as implementações e testes, descrito a seguir.

5.1 Apresentação do estudo de caso

O ponto de partida do trabalho aqui apresentado foi o projeto descrito em Lima et al (2004), com vistas à construção de um protótipo de ferramenta para simulação de exame de punção de mama.

Lima et al (2004) cita que o câncer de mama é o tipo de câncer mais freqüente em incidência e mortalidade em pessoas do sexo feminino, sendo a punção aspirativa por agulha fina um dos métodos de diagnóstico mais comum na comunidade médica para distinção entre malignidade e benignidade de tumores. Por se tratar de um procedimento médico que exige destreza por parte do profissional, o protótipo visa contribuir para o treinamento de alunos de

medicina e recém-formados, a fim de evitar incisões incorretas, coleta de material errôneo ou insuficiente e até mesmo desconforto à paciente.

O protótipo foi desenvolvido na linguagem Java em conjunto com a biblioteca Java 3D, tendo o MySQL (MYSQL, 2004) como suporte para banco de dados e imagens, e o aplicativo 3DStudio (DISCREET, 2004) para modelagem dos objetos 3D utilizados na aplicação.

Na Figura 5-1 é apresentado o AV disponibilizado pelo protótipo. À esquerda podem ser observadas imagens de mamogramas digitalizados (raio-X de mama) e à direita são disponibilizados objetos tridimensionais (seringa e mama) para simular o ambiente real utilizado para treinamento de estudantes. Já a Figura 5-2, apresenta os objetos 3D que representam a mama e a seringa, respectivamente, no AV do protótipo.

Segundo descrito em Lima et al (2004), o protótipo é um sistema que apresenta característica de Realidade Virtual em uma plataforma convencional do tipo PC, com as interações disponibilizadas pelo mouse e a visualização em monitor de vídeo. Pela ausência de equipamentos não convencionais, como dispositivo háptico e óculos estereoscópicos, não há imersão por parte do usuário.

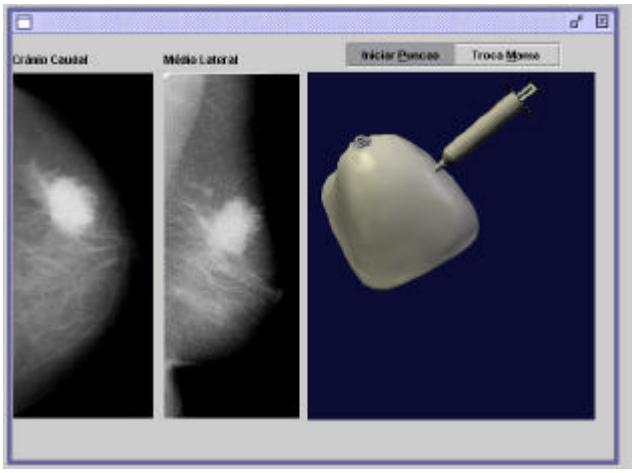


FIGURA 5-1: AV DO PROTÓTIPO DE FERRAMENTA PARA SIMULAÇÃO DE EXAME DE PUNÇÃO DE MAMA (LIMA ET AL, 2004).

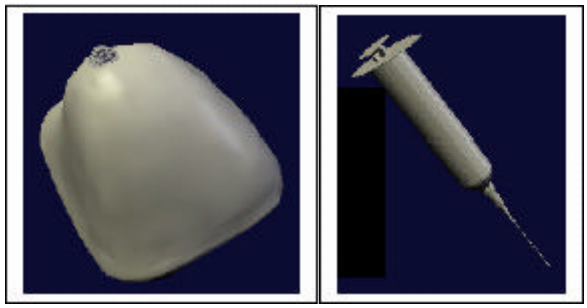


FIGURA 5-2: OBJETOS SINTÉTICOS 3D QUE REPRESENTAM A MAMA E A SERINGA NO AV (LIMA ET AL, 2004).

Os principais módulos do protótipo são: Módulo de Banco de Dados e Módulo de Exame de Punção. O Módulo de Banco de Dados é responsável pelo cadastro e armazenamento de dados e imagens de raio-X de pacientes reais. Já o de Exame de Punção, é responsável pela realização do exame de punção a partir dos dados armazenados.

Conforme citado em Lima et al (2004), por haver determinados estágios da execução do exame em que somente partes da seringa são manipuladas, o modelo que representa a seringa foi construído em partes e nelas, na implementação, foram atribuídos controles específicos. Por exemplo, como em um exame de punção real o médico deve atingir o nódulo e colher o material celular do mesmo, funções de detecção de colisão foram implementadas no objeto virtual que representa a agulha. Já para o movimento de "vai-e-vem" do êmbolo no momento da aspiração do material celular, foi preciso implementar rotinas específicas para o movimento e posicionamento do mesmo no AV.

Para melhorar o protótipo, Lima et al (2004) citam como implementações futuras: a inserção de dispositivo háptico que permitiria ao usuário determinar a força necessária para aplicar na seringa no momento da incisão, bem como, sentir o rompimento da pele e estruturas da mama à medida que a seringa "caminha" de encontro ao nódulo; o uso de luvas de dados, que permitiriam ao usuário obter a sensação de tato na mama para fixar o nódulo; a inclusão de deformação dinâmica através da triangulação dos objetos tridimensionais

acontecendo em tempo de execução e de acordo com o toque da luva na mama e a segmentação da imagem de raio-X para a construção dinâmica do nódulo tridimensional.

5.2 Considerações sobre J3D e WTK

Além de ter um alto nível de abstração quanto à programação gráfica, a biblioteca J3D foi escolhida pelo fato do protótipo de ferramenta descrito em Lima et al (2004) ter sido implementado na linguagem Java em conjunto com esta biblioteca, mas sem refinamento do método de detecção de colisão. O WTK, por sua vez, foi selecionado pelo alto nível de abstração, além das vantagens como boa documentação, suporte e portabilidade, uma vez que suas funções são implementadas na linguagem C em conjunto com a biblioteca OpenGL.

Para obter uma análise sobre os métodos oferecidos pela J3D e WTK para detecção de colisão, vários testes e implementações foram realizados. Este estudo permitiu verificar as vantagens e desvantagens de cada método, além do desempenho da aplicação baseada em *frames* por segundo (FPS), isto é, na taxa média de quadros de imagem por segundo que ela consegue apresentar.

Um fator importante para os testes realizados, foi o fato das tecnologias de *software* selecionadas não oferecerem métodos específicos para tratamento de colisão entre os vários objetos da cena 3D. Como na aplicação descrita em Lima et al (2004) a seringa é composta por partes como o êmbolo e a agulha, o objeto referente ao êmbolo teve seu *status* de colisão desabilitado, sendo avaliada somente a detecção de colisão entre o objeto da agulha e o objeto da mama.

Para conhecer a complexidade dos objetos modelados foi verificada, por funções específicas do WTK, a quantidade de polígonos (triângulos) que formam cada objeto. Como resultado, o objeto agulha é formado por 893 polígonos e a mama por 5278 polígonos.

A plataforma computacional adotada para execução dos testes e desenvolvimento das implementações, foi composta por um PC Pentium III 500 MHz, 512 Mb de memória RAM, placa de vídeo NVIDIA com 32 Mb e Sistema Operacional Windows XP Professional. Devido à atual indisponibilidade de dispositivos não convencionais, como luvas e dispositivos hápticos, o projeto foi desenvolvido baseando-se em dispositivos de entrada comuns, neste caso, teclado e *mouse*.

5.3 Detecção de colisão com J3D

A J3D possui métodos que permitem detectar quando um objeto entra, está ou sai de uma colisão com outro objeto. Os métodos disponíveis detectam colisão de forma aproximada e rápida a partir do envolvimento do objeto por volumes limites e de maneira mais precisa e lenta pela geometria do objeto. Como os métodos disponíveis verificam colisão entre dois objetos, surgem dificuldades para implementar situações de colisões em n-objetos, isto é, se 3 (três) ou mais objetos estão colidindo ao mesmo tempo juntos (J3D.ORG, 2004).

Assim, o estudo aqui apresentado teve o objetivo de, num primeiro momento, quantificar os métodos disponíveis na J3D por meio do uso de volumes limites (USE_BOUNDS) e da geometria dos objetos (USE_GEOMETRY). Os FPS foram coletados tanto utilizando a versão da J3D baseada na OpenGL, quanto na baseada no DirectX, sem necessidade de alteração do código-fonte.

Para realizar os testes de colisão entre o objeto da agulha e o objeto da mama, a classe implementada (CollisionDetector) foi associada ao comportamento do objeto da agulha. Este trecho do código-fonte pode ser observado na Figura 5-3.

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

FIGURA 5-3: ASSOCIAÇÃO DA CLASSE PARA DETECÇÃO DE COLISÃO AO OBJETO AGULHA.

Já a parametrização sobre o tipo de método que deveria ser aplicado para detecção de colisão entre objetos, neste caso, USE_BOUNDS para uso de volumes limites ou USE_GEOMETRY para uso da geometria dos objetos, foi definida na classe CollisionDetector. A Figura 5-4 apresenta o trecho do código-fonte que define esta parametrização.

```
public void initialize(){
  mouseEvents = new WakeupCriterion[3];
  mouseEvents[0] = new WakeupOnCollisionEntry(s, WakeupOnCollisionEntry.USE_BOUNDS);
  mouseEvents[1] = new WakeupOnCollisionExit(s, WakeupOnCollisionExit.USE_BOUNDS);
  mouseEvents[2] = new WakeupOnCollisionMovement(s, WakeupOnCollisionMovement.USE_BOUNDS);
  mouseCriterion = new WakeupOr(mouseEvents);
  this.wakeupOn(mouseCriterion);
}
```

FIGURA 5-4: PARAMETRIZAÇÃO SOBRE O MÉTODO DE DETECÇÃO DE COLISÃO A SER USADO PELA J3D.

Como a J3D não possuía nenhuma classe definida para coleta dos FPS, foi necessário criar uma classe específica para esta tarefa e, após, fazer a associação ao objeto representando a seringa, uma vez que o objeto da mama é estático. A Figura 5-5 mostra trecho do código-fonte da classe FramesPorSegundo para cálculo da taxa média de quadros de imagem por segundo.

```
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;
                        System.out.println((double)IntervaloRelatorio / (Intervalo/1000.00));
                  }
                  TempoInicial = System.currentTimeMillis();
            }
      }
      wakeupOn (Condicao);
}
```

FIGURA 5-5: TRECHO DA CLASSE FRAMESPORSEGUNDO PARA CÁLCULO DOS FPS NA J3D.

5.4 Detecção de colisão com WTK

O WTK oferece diversos métodos para detecção de colisão entre objetos de um AV. Estes métodos podem verificar interação entre um polígono que compõe um objeto e qualquer outro polígono que faz parte do mesmo objeto ou de outro especificado, entre um polígono e qualquer parte do volume limite que envolve um objeto específico, entre quaisquer polígonos nos objetos especificados e entre dois objetos especificados baseados em seus volumes limites (SENSE8, 2003).

Um fator importante no processo de implementação da coleta de FPS no WTK foi o fato deste ter uma função específica para este fim (WTuniverse_framerate()), a qual possibilitou um desenvolvimento mais rápido, uma vez que na J3D foi preciso implementar uma classe específica para este processo.

Os seguintes métodos para detecção de colisão entre dois objetos foram implementados e testados:

- WTnodepath_intersectbbox – testa a intersecção entre dois objetos baseado nos seus volumes limites.

- WTpoly_intersectnode - testa a intersecção de um polígono do objeto informado com qualquer polígono de outro objeto.
- WTnodepath_intersectpoly - testa a intersecção de qualquer polígono entre dois objetos.
- WTpoly_intersectpolygon - testa a intersecção entre dois polígonos.

A Figura 5-6 apresenta a utilização de cada método para detecção de colisão entre os objetos do AV no WTK. Cada método foi implementado e testado individualmente a fim de fazer a coleta dos FPS. Esta etapa foi importante para a geração do gráfico comparativo do desempenho de cada método.

```
/* Usando o WTnodepath_intersectbbox */

if (WTnodepath_intersectbbox(nodepathagulha, nodepathamama)){
        WTmessage("Colisão detectada - FPS %2.2f\n", WTuniverse_framerate());
        colisao = TRUE;
}


/* Usando o WTpoly_intersectnode */

for (a=0; a<NUMPOLYSAGULHA; a++){
        if (WTpoly_intersectnode(polyagulha[a], nodepathagulha, nodepathmama)){
                WTmessage("Colisão detectada - FPS %2.2f\n", WTuniverse_framerate());
                a = NUMPOLYSAGULHA;
                colisao = TRUE;
        }
}


/* Usando o WTnodepath_intersectpoly */

if (WTnodepath_intersectpoly(nodepathagulha, nodepathmama)){
        WTmessage("Colisão detectada - FPS %2.2f\n", WTuniverse_framerate());
        colisao = TRUE;
}


/* Usando o WTpoly_intersectpolygon */

for (a=0; a<NUMPOLYSAGULHA; a++){
        if (WTpoly_intersectpolygon(polymama[m], nodepathmama, polyagulha[a], nodepathagulha)){
                WTmessage("Colisão detectada - FPS %2.2f\n", WTuniverse_framerate());
                m = NUMPOLYSMAMA;
                a = NUMPOLYSAGULHA;
                colisao = TRUE;
        }
}
```

FIGURA 5-6: USO DOS MÉTODOS PARA DETECÇÃO DE COLISÃO NO WTK.

5.5 Métodos para refinamento

Segundo Garcia-Alonso et al (1994), tendo em vista que a execução de algoritmos mais refinados pode causar um custo de processamento maior, soluções mais simples, como o

uso de volumes limites, normalmente são adotadas. Neste caso, se nos testes de intersecção não houver nenhuma sobreposição, não haverá colisão, mas se houver, o risco existirá e outros métodos mais refinados e custosos poderão ser aplicados.

Assim, a partir dos estudos realizados com os métodos oferecidos pela J3D e pelo WTK, passou-se a identificar possíveis refinamentos do processo de detecção de colisão.

5.5.1 Refinamento com J3D

Além do fato da J3D não possuir método para detecção de colisão entre n-objetos, outra dificuldade encontrada foi a não atualização das coordenadas dos vértices dos objetos a partir de uma transformação (rotação, translação ou mudança de escala), situação constatada por classes específicas para tratamento das geometrias dos objetos. Os objetos utilizados no projeto descrito em Lima et al (2004) foram todos modelados pelo software 3DStudio e exportados para o tipo *Wavefront Object* (.obj), os quais são carregados e posicionados no AV a partir de uma classe específica desta biblioteca.

Uma primeira solução, simples e rápida para o refinamento da detecção de colisão, foi calcular a distância euclidiana entre as coordenadas centrais de cada objeto (mama e agulha) e informar se estavam efetivamente em colisão a partir da comparação com um limiar mínimo. A distância euclidiana (CÂMARA et al, 2004) é dada pela equação (1), onde CA e CM são as coordenadas centrais da agulha e da mama, respectivamente.

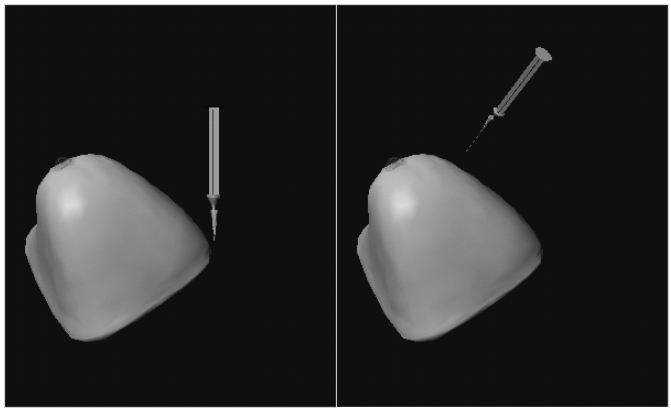
$$\mathbf{d(CA,CM)=}\sqrt{(CA_x - CM_x)^2 + (CA_y - CM_y)^2 + (CA_z - CM_z)^2} \qquad (1)$$

O trecho do código-fonte da classe CollisionDetector responsável por este cálculo, pode ser visto na Figura 5-7.

```
//Calcula distância euclidiana entre as coordenadas centrais dos objetos
double dist = Math.sqrt(Math.pow((v3fAgulha.x - v3fMama.x),2)+Math.pow((v3fAgulha.y - v3fMama.y),2)+
                        Math.pow((v3fAgulha.z - v3fMama.z),2));
```

FIGURA 5-7: CÁLCULO DA DISTÂNCIA EUCLIDIANA NA J3D.

Por se tratar de objetos com geometria complexa, esta solução mostrou-se ineficiente em determinadas localizações do AV, mesmo alterando o limiar. A Figura 5-8 mostra dois momentos da aplicação deste algoritmo indicando: (a) uma aproximação mais precisa e (b) uma aproximação imprecisa, na qual foi detectada uma colisão sem realmente esta ter ocorrido.



a) Aproximação mais precisa b) Aproximação imprecisa

FIGURA 5-8: MÉTODO DA DISTÂNCIA EUCLIDIANA NA J3D.

Uma alternativa para aumentar a acurácia deste algoritmo foi verificar se a coordenada central da agulha estava sobre algum polígono (triângulo) pertencente à superfície do objeto da mama. Esta verificação foi possível a partir do cálculo da equação do plano (CÂMARA et al, 2004) dada pela equação (3), onde:

- **n** é o vetor normal em relação ao plano (vetor perpendicular ao plano **q**, neste caso, o polígono pertencente à superfície do objeto da mama) dada pela equação (2);
- **p** é o ponto a ser verificado (coordenada central da agulha, considerando as posições **x**,**y** e **z**) e
- **d = -n.p**, a distância de **p** em relação ao plano.

Caso a distância **d** seja menor ou igual a zero, o ponto **p** está do lado de dentro ou na superfície do plano **q**, respectivamente. A Figura 5-9 mostra o trecho do código-fonte da classe CollisionDetector responsável pelo cálculo em J3D.

$$\mathbf{n = (q1-q0) \times (q2-q1)} \qquad (2)$$

$$\mathbf{f(p) = n.p+d = 0} \qquad (3)$$

Mesmo tendo que percorrer os vértices que compõem o objeto da mama para encontrar o polígono sob o objeto da agulha, o processo não prejudicou o desempenho da aplicação e conseguiu aumentar a precisão da detecção de colisão. Testes também revelaram que evitar a transformação (rotação e translação) do objeto mama para posicionamento no AV pode aumentar a eficiência dos métodos aplicados. Um exemplo de transformação pode ser observado na Figura 5-8, enquanto a Figura 5-10 mostra uma cena na qual não houve transformações.

```
for (int i=0; i<gaMama.getVertexCount(); i+=3){

      Point3f[] u = new Point3f[3];
      u[0] = vMama[i];
      u[1] = vMama[i+1];
      u[2] = vMama[i+2];

      //Cálculo da Normal: N=(q1-q0)x(q2-q1)
      Point3f E1 = new Point3f();
      Point3f E2 = new Point3f();
      Point3f N = new Point3f();
      E1.sub(u[0], u[1]);
      E2.sub(u[1], u[2]);
      N.x = E1.y*E2.z-E1.z*E2.y;
      N.y = E1.z*E2.x-E1.x*E2.z;
      N.z = E1.x*E2.y-E1.y*E2.x;

      //Cálculo da equação do plano: d=-n.p
      float d = (-1)* ((N.x*v3fAgulha.x)+(N.y*v3fAgulha.y)+(N.z*v3fAgulha.z));

      if (d<=0.00f){
            System.out.println("Em colisão. Distância calculada: "+d);
            i = gaMama.getVertexCount();
      }

}
```

FIGURA 5-9: CÁLCULO DA EQUAÇÃO DO PLANO NA J3D.

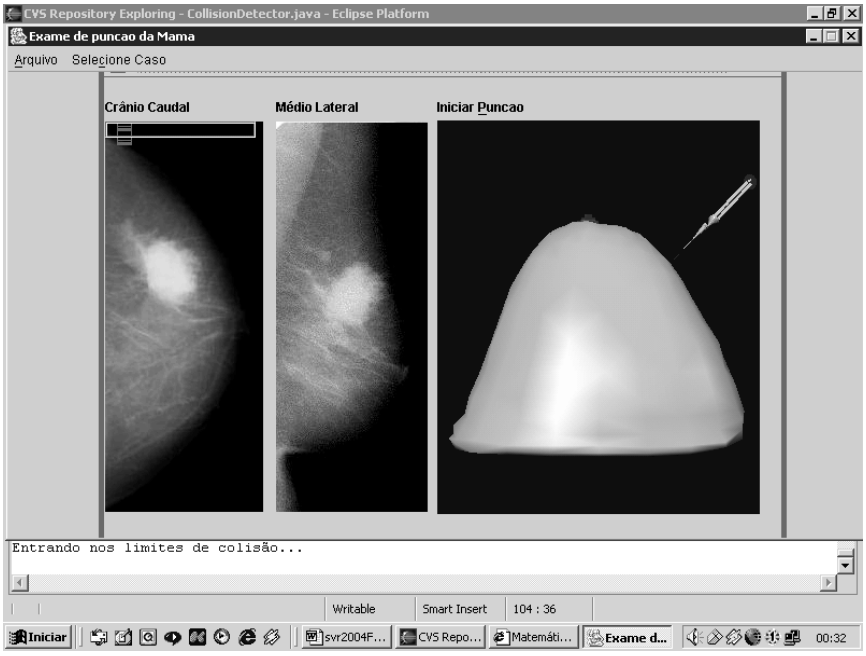


FIGURA 5-10. REFINAMENTO SEM TRANSFORMAÇÃO DO OBJETO MAMA NA J3D.

Também foram realizados testes para avaliar o uso da biblioteca V-CLIP (V-CLIP, 2003) em conjunto com a linguagem Java e a biblioteca J3D para uma detecção de colisão mais precisa.

A biblioteca V-CLIP foi projetada para permitir detectar colisão entre poliedros convexos, mas conforme a verificação por classes específicas desta biblioteca, os objetos 3D que representam a agulha e a mama foram considerados não convexos por não satisfazerem os critérios da Fórmula de Euler, resultado que inviabilizou a continuidade dos testes. A Fórmula de Euler é dada ela equação (4), onde **V** é o número de vértices, **F** corresponde ao número de faces e **A** é o número de arestas.

$$V+F=A+2 \qquad (4)$$

5.5.2 Refinamento com WTK

Assim como a J3D, o WTK também não possui métodos específicos para detecção de colisão entre n-objetos. Entretanto, é possível obter os vértices atualizados dos objetos após uma transformação por funções específicas desta tecnologia. Por outro lado, não foi possível trabalhar somente com arquivos do tipo Wavefront Object (.obj). Neste caso, somente o objeto referente à mama foi carregado para o AV a partir de um arquivo .obj. A agulha, por sua vez, partiu de um arquivo com extensão 3DS, tipo de arquivo padrão do 3DStudio. Na documentação existente não foi encontrado motivo, bem como solução, para este problema.

Assim como na coleta de FPS, o WTK também possui funções já definidas para cálculo da distância euclidiana (WTp3_distance) e para o cálculo da normal da equação do plano (WTpoly_getnormal), permitindo acelerar a implementação dos métodos de refinamento.

Quanto ao número de linhas de programação, na Figura 5-11 é possível notar uma redução considerável no desenvolvimento da equação do plano em relação à J3D.

```
/*cálculo da equação do plano -n.p*/
WTpoly_getnormal(polymama[m], normal);
distancia = (-1) * WTp3_dot(pA, normal);
```

FIGURA 5-11: CÁLCULO DA EQUAÇÃO DO PLANO NO WTK.

Quanto à precisão dos métodos implementados, tanto o método da distância euclidiana quanto o da equação do plano, demonstraram resultados parecidos àqueles implementados com J3D. Após encontrar os limiares mais adequados para detecção de colisão, o método da equação do plano aumentou a acurácia do processo de detecção de colisão sem prejudicar o desempenho do sistema. A Figura 5-12 mostra o momento em que o método da distância euclidiana detectou a colisão e, logo em seguida, o método da equação do plano confirmou o contato dos objetos.

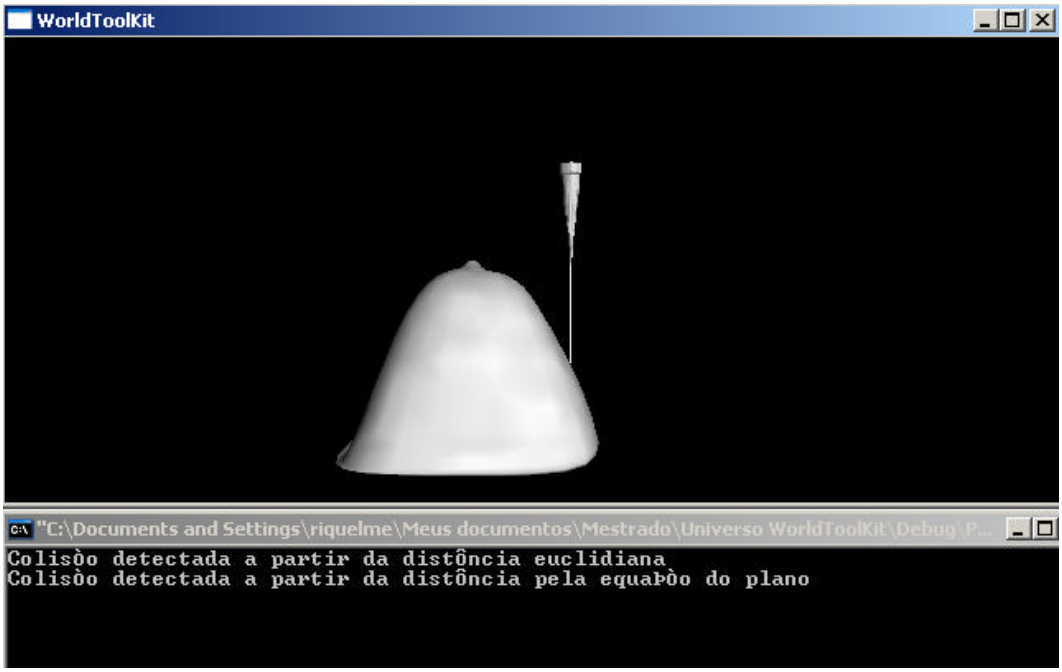


FIGURA 5-12: REFINAMENTOS NO WTK.

6. RESULTADOS E DISCUSSÕES

Como apresentado, há vários projetos baseados em RV em fase de desenvolvimento ou teste. No capítulo 2, diversos projetos foram apresentados e, pela Tabela 2-1, podemos identificar as tecnologias de *hardware* e *software* utilizadas em cada projeto. Entretanto, mesmo havendo projetos direcionados a uma mesma área de aplicação e utilizando a mesma tecnologia, o problema abordado não era o mesmo.

Diante deste fato, é difícil definir exatamente qual a tecnologia de *hardware* e *software* poderia apresentar melhores resultados aplicados a uma determinada área e problema. O mesmo ocorre com o problema da detecção de colisão entre objetos em um AV, onde existem diversas abordagens e métodos que podem ser aplicados. Neste caso, somente é possível afirmar que os métodos simples são imprecisos e os mais refinados causam um alto custo computacional, como abordado em Garcia-Alonso et al (1994) e comprovado pelos estudos realizados no capítulo 5.

Assim, neste capítulo, será apresentada uma análise comparativa dos métodos oferecidos pela J3D e pelo WTK em relação ao método da equação do plano desenvolvido. Esta análise é baseada na precisão e no desempenho da aplicação de acordo com a quantidade de FPS, onde a plataforma de execução foi a mesma considerada no capítulo 5, assim como o projeto descrito.

6.1 Testes iniciais de precisão e desempenho

Durante a coleta dos FPS na J3D, os testes revelaram que a utilização da geometria (USE_GEOMETRY) para detecção de colisão reduzia drasticamente o desempenho da aplicação, tanto na versão baseada em OpenGL quanto na baseada no DirectX, conforme mostram as Figuras 6-1 e 6-2.

Os dados foram coletados a cada taxa de FPS calculada pela classe FramesPorSegundo. Estes momentos correspondem aos intervalos (título do eixo X) descritos nos gráficos. A diminuição da taxa de FPS em cada método mostra o momento de interação do usuário com o AV, isto é, o usuário movendo a agulha em direção à mama. Na detecção de colisão com USE_GEOMETRY, quando a taxa de FPS ficou muito baixa, houve um travamento temporário em todo o sistema.

Além do desempenho quantificado, foi realizada uma avaliação visual a respeito da precisão, tendo sido percebido que a utilização do método dos volumes limites serve apenas para indicar uma possível colisão. Este fato pode ser comprovado a partir da observação da Figura 6-3, que representa um momento da interação entre os dois objetos sintéticos modelados (representando uma seringa e uma mama), no qual a J3D indicou que havia uma colisão, quando é possível verificar visualmente que esta ainda não existia. Da mesma forma, percebeu-se que o uso das geometrias dos objetos também era inviável para o tipo de aplicação descrita em Lima et al (2004), visto que demandava um alto custo computacional.

Para uma avaliação mais adequada destes métodos, isto é, não visual, poderia ter sido implementada uma rotina que verificasse se algum ponto do objeto seringa pertencia, ou não, ao objeto mama no momento da colisão. Entretanto, para este procedimento seria necessário percorrer todos os pontos de ambos os objetos, o que causaria um alto custo computacional, como foi o caso do método usando o parâmetro USE_GEOMETRY.

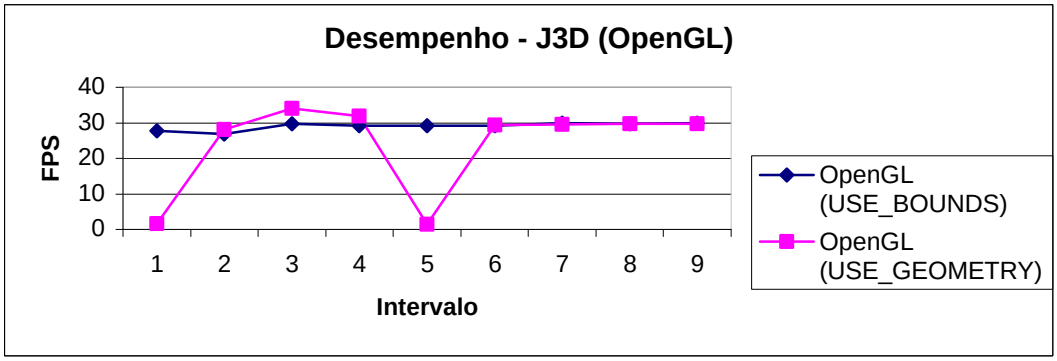


FIGURA 6-1: DESEMPENHO DA J3D BASEADA NA OPENGL.

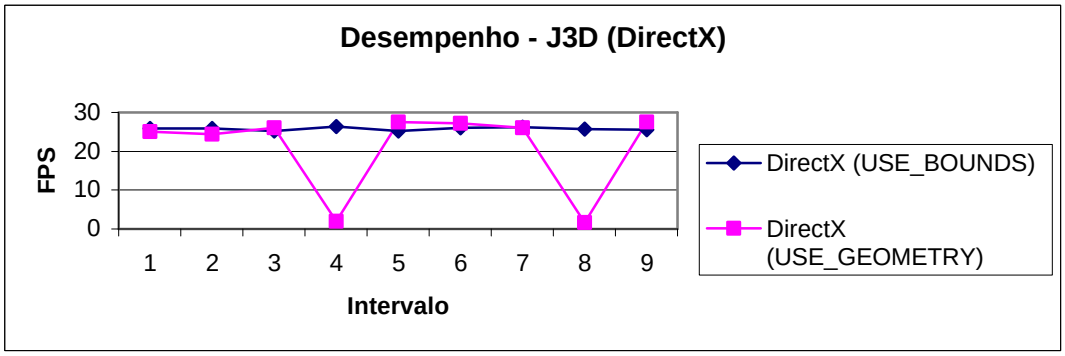


FIGURA 6-2: DESEMPENHO DA J3D BASEADA NO DIRECTX.

Como na J3D, os métodos para detecção de colisão baseados na geometria dos objetos no WTK causaram uma queda considerável de FPS nos momentos de interação do usuário com o AV. Os resultados podem ser observados na Figura 6-4.

A avaliação da precisão do método considerando somente os volumes limites que envolvem os objetos (WTnodepath_intersectbbox) também foi insatisfatória, uma vez que indicou uma colisão, quando na verdade, visualmente, esta não existia (Figura 6-5). Além disso, pelo alto custo computacional, percebeu-se que o uso das geometrias dos objetos também era desaconselhável para o tipo de aplicação discutida em Lima et al (2004).

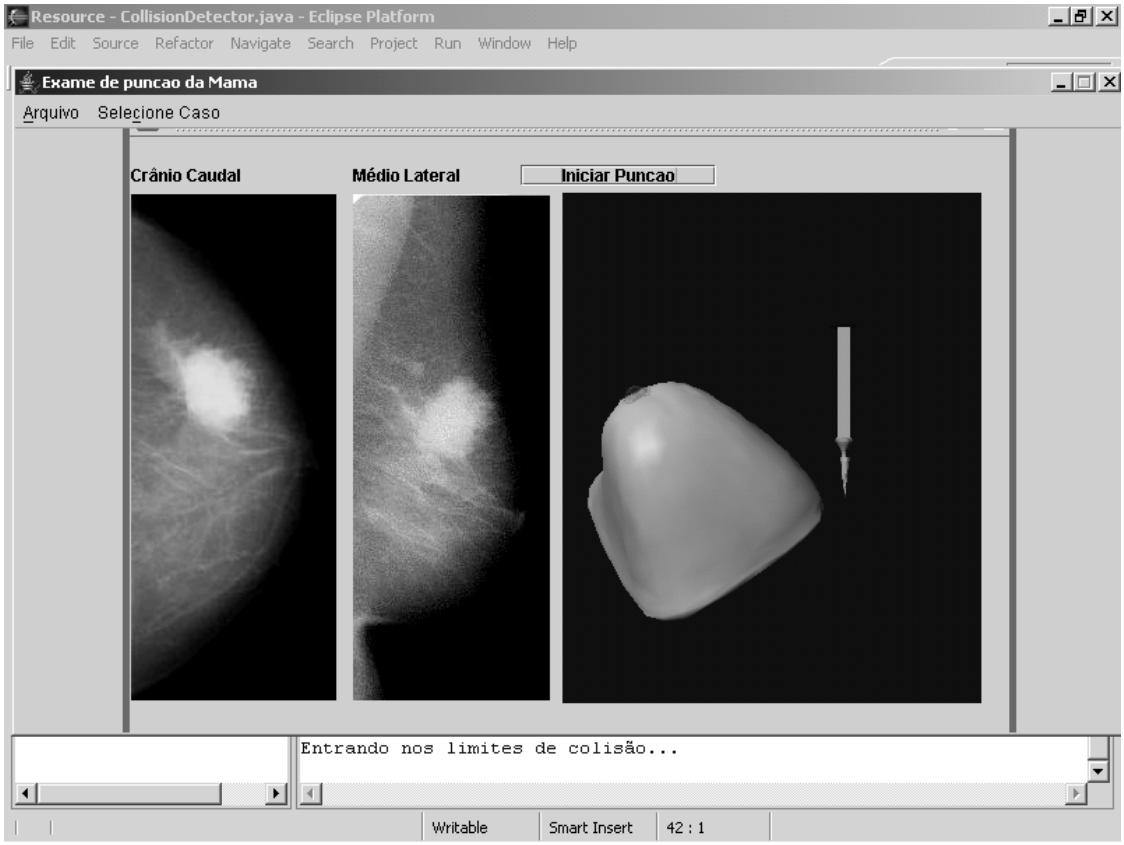


FIGURA 6-3: DETECÇÃO DE COLISÃO COM USO DE VOLUMES LIMITES NA J3D.

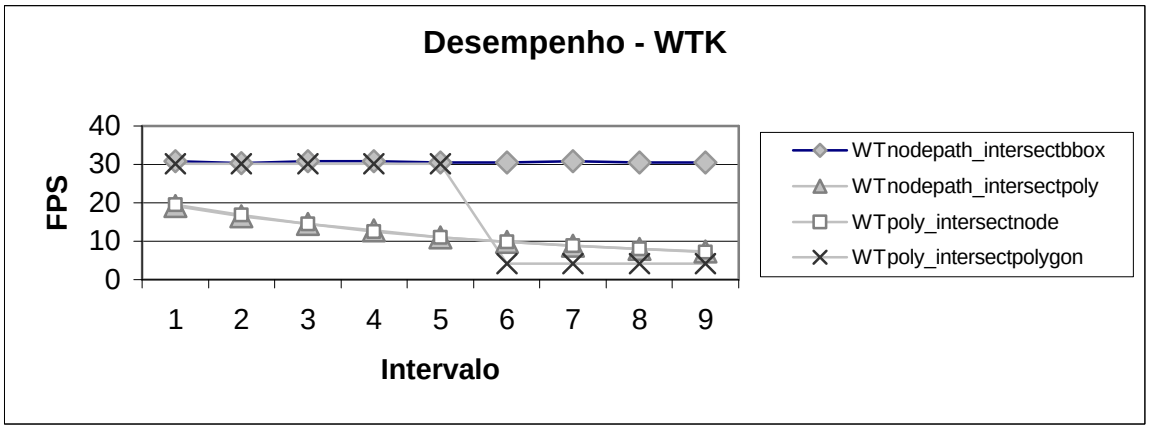


FIGURA 6-4: DESEMPENHO DO WTK.

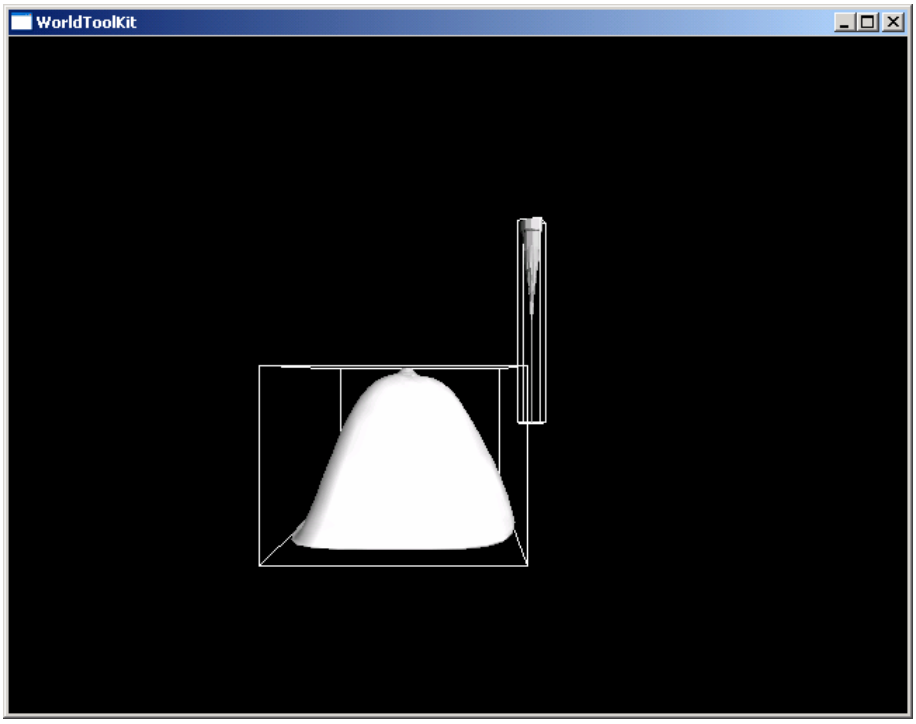


FIGURA 6-5: DETECÇÃO DE COLISÃO COM USO DE VOLUMES LIMITES NO WTK.

6.2 Definição de limiares

O refinamento da detecção de colisão baseado na distância euclidiana necessitava de um limiar mínimo para verificar se os objetos sintéticos referentes à agulha e à mama estavam, ou não, em colisão. Para tal, foram coletadas diversas distâncias a fim de prover o limiar mais adequado para a aplicação descrita em Lima et al (2004).

Na implementação do refinamento na J3D, o valor ideal do limiar foi 0.8. Já no WTK, este valor foi de 20.0. A principal diferença está no tipo de dado retornado pelos cálculos realizados, onde a J3D retornou *double* e o método WTp3_distance do WTK trabalha com o tipo *float*. Infelizmente, por falta de detalhamento da documentação de ambas tecnologias, não foi possível definir a unidade de medida usada nos AVs gerados.

Outra dificuldade encontrada foi o fato da obtenção destes limiares ter sido de forma manual, pois é praticamente impossível executar manualmente o procedimento de deslocamento do objeto agulha até o objeto mama obedecendo a mesma trajetória de um procedimento anteriormente realizado. Uma forma de resolver esta questão futuramente é executar as movimentações de forma automatizada, por meio, por exemplo, de implementação de rotinas de animações pré-definidas.

6.3 Método da equação do plano comparado a J3D

O método mais preciso oferecido pela J3D é o USE_GEOMETRY, parâmetro informado à classe de tratamento de colisão que habilita os testes de intersecção baseado na geometria dos objetos e não em seus volumes limites (USE_BOUNDS).

Nos testes realizados, o USE_GEOMETRY da J3D apresentou uma taxa abaixo de 2 FPS quando o usuário estava interagindo com o AV. Nesta situação, causando o travamento temporário de todo o sistema. Já o método da equação do plano conseguiu aumentar a precisão sem prejudicar o desempenho. Os dados coletados mostraram uma taxa de FPS acima de 18 no momento da interação do usuário, tanto na versão baseada na OpenGL quanto na baseada no DirectX. As Figuras 6-6 e 6-7 mostram o desempenho dos dois métodos nas versões citadas.

Comparado ao método mais simples e menos preciso da J3D, o qual é baseado nos volumes limites que envolvem os objetos (USE_BOUNDS), o método da equação do plano teve taxas de FPS mais baixas. Isto se deve ao fato que é necessário percorrer os vértices do

objeto da mama para encontrar o polígono sob o objeto da agulha. As Figuras 6-8 e 6-9 apresentam os resultados obtidos.

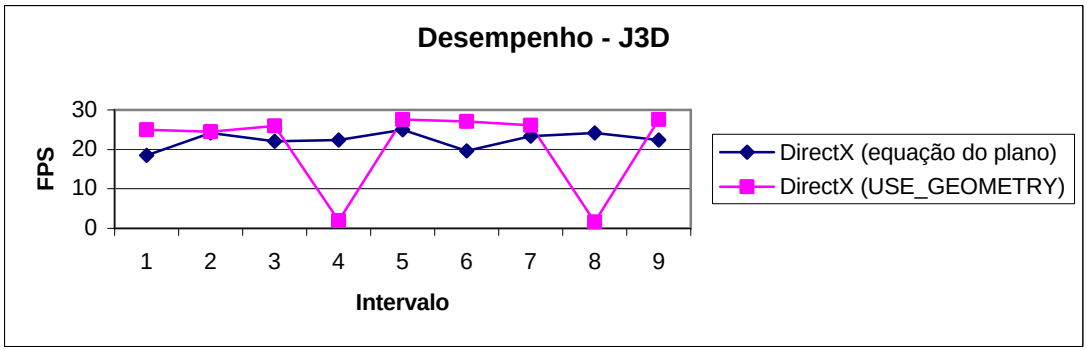


FIGURA 6-6: MÉTODO DA EQUAÇÃO DO PLANO VS J3D (DIRECTX – USE_GEOMETRY).

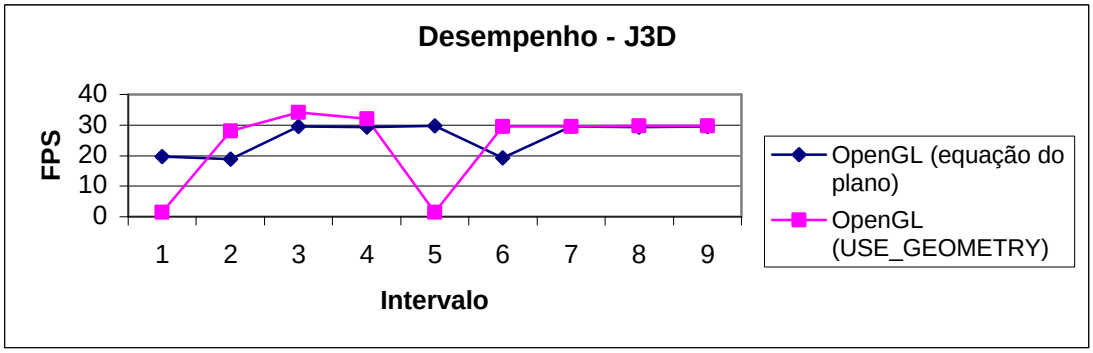


FIGURA 6-7: MÉTODO DA EQUAÇÃO DO PLANO VS J3D (OPENGL – USE_GEOMETRY).

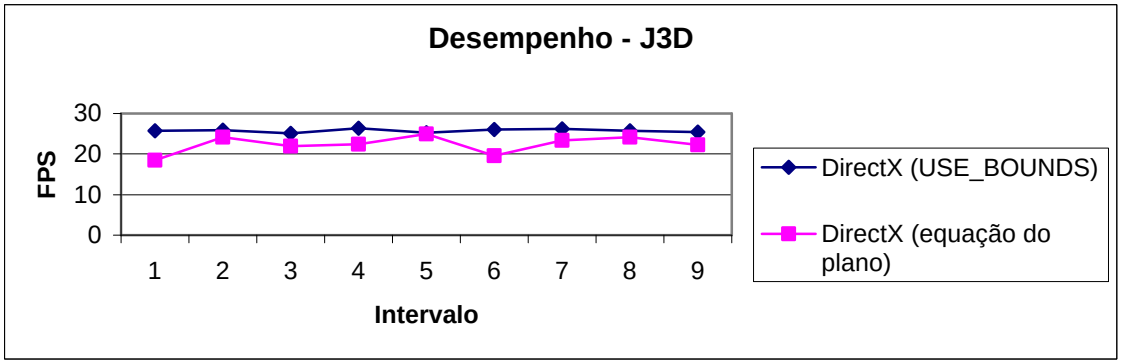


FIGURA 6-8: MÉTODO DA EQUAÇÃO DO PLANO VS J3D (DIRECTX – USE_BOUNDS).

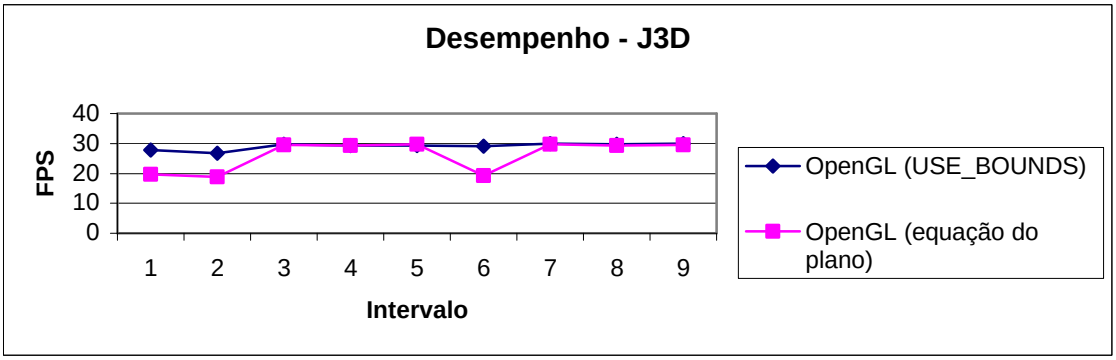


FIGURA 6-9: MÉTODO DA EQUAÇÃO DO PLANO VS J3D (OPENGL – USE_BOUNDS).

6.4 Método da equação do plano comparado ao WTK

Como observado, o WTK oferece diversos métodos para verificação de intersecção entre dois objetos baseado em suas geometrias, além de um método rápido para detecção de colisão por meio de volumes limites. Mas idêntico a J3D, estes métodos causam um alto custo computacional, situação que compromete o desempenho de toda a aplicação. Esta situação pôde ser observada na Figura 6-4, onde a taxa de FPS para estes métodos vai gradativamente caindo conforme há interação do usuário.

De acordo com os dados coletados durante a execução do método da equação do plano, o desempenho da aplicação manteve-se estável com uma taxa acima de 19 FPS. A Figura 6-10 mostra os intervalos coletados em relação aos outros métodos mais precisos existentes no WTK.

Conforme o gráfico gerado, é possível perceber que o método WTpoly_intersectpolygon difere dos outros métodos oferecidos pelo WTK. Enquanto os métodos WTpoly_intersectnode e WTnodepath_intersectpoly vão gradativamente reduzindo o desempenho da aplicação, isto é, reduzindo a taxa de FPS conforme a interação do usuário no AV, o WTpoly_intersectpolygon só afeta a aplicação quando é efetivamente

chamado. Na documentação disponível do WTK não havia informação que explicasse tal fato, mas pelos testes realizados pode-se afirmar que esta situação ocorra devido ao tempo de execução de cada método e da forma que foram implementados.

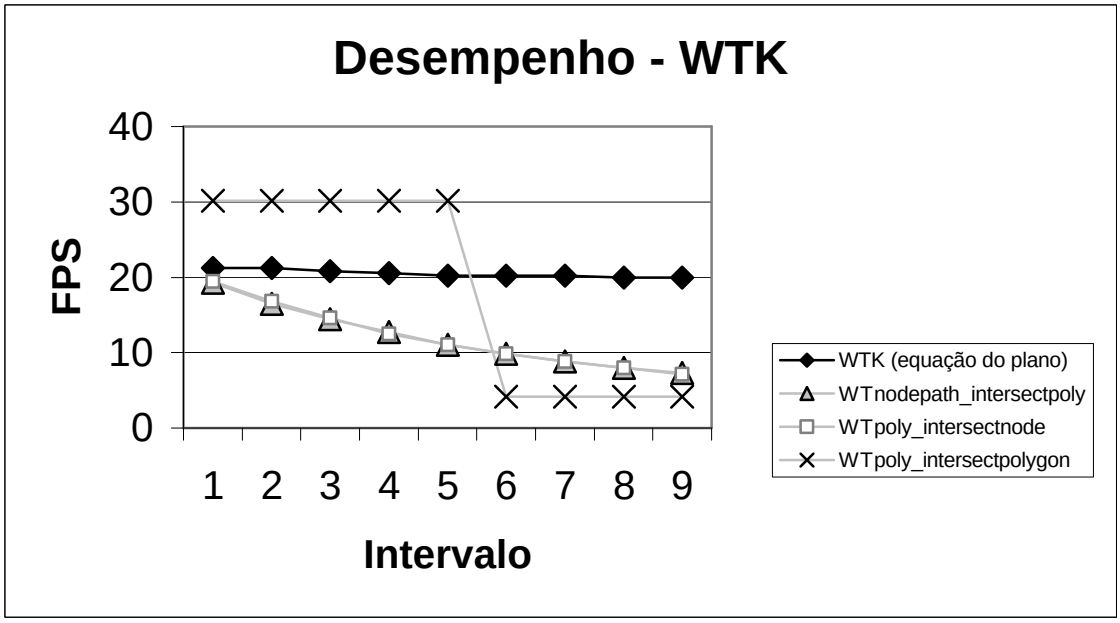


FIGURA 6-10: MÉTODO DA EQUAÇÃO DO PLANO VS WTK (MÉTODOS BASEADOS NA GEOMETRIA DOS OBJETOS).

Os métodos WTpoly_intersectnode e WTnodepath_intersectpoly varrem, pelo menos, um dos objetos em seu todo para detectar colisão ou não. O WTpoly_intersectpolygon, por sua vez, testa a intersecção entre dois polígonos informados, tendo um retorno mais rápido do que os métodos anteriores. Entretanto, para o tipo da aplicação implementada, era importante checar se um determinado polígono do objeto agulha estava, ou não, em colisão com algum polígono do objeto mama, motivo pelo qual houve a queda dos FPS.

Assim como na J3D, o método da equação do plano obteve taxas de FPS mais baixas que o método do WTK baseado nos volumes limites dos objetos

(WTnodepath_intersectbbox), mas tal fato não comprometeu o desempenho da aplicação. Esta situação pode ser observada na Figura 6-11.

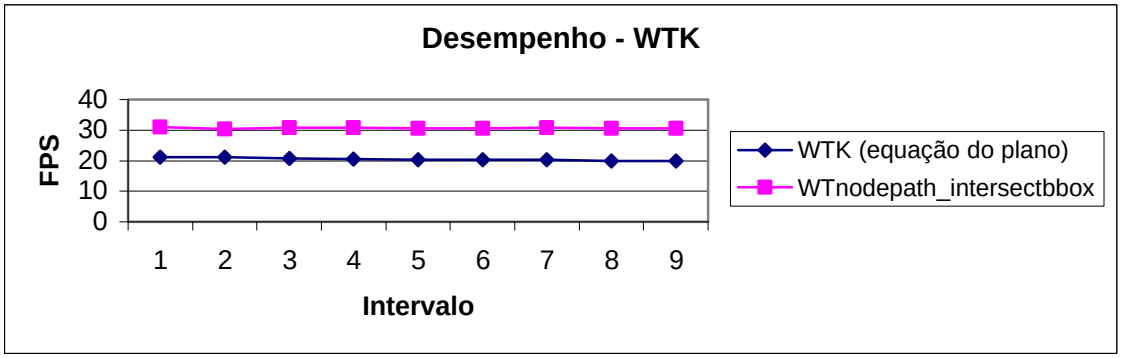


FIGURA 6-11: MÉTODO DA EQUAÇÃO DO PLANO VS WTK (VOLUMES LIMITES).

6.5 J3D comparada ao WTK

Percebe-se que na comunidade de RV ainda pairam muitas dúvidas em relação à escolha de tecnologia para a realização de projetos. O fator custo é determinante principalmente quando o problema envolve a utilização de *hardware* não convencional. No entanto, fatores como desempenho, facilidades para o desenvolvedor e para o usuário, entre outros, podem ser importantes na seleção de ferramentas para implementação. Observa-se, entretanto, que praticamente não há trabalhos de pesquisa para analisar tais fatores. Salienta-se, também, que trabalhos nesta linha devem ter uniformidade em relação ao problema estudado.

Neste sentido, uma análise comparativa entre a J3D e o WTK, considerando os métodos discutidos e o desempenho da aplicação durante os testes de execução, podem auxiliar futuros projetos. Outro fator que deve ser considerado para esta discussão é o fato da J3D ter duas versões disponíveis, uma baseada na OpenGL e outra no DirectX.

Comparando as versões da J3D, o único fator considerado foi o desempenho, uma vez que o código implementado não precisou sofrer alteração para a execução. Assim, analisando as taxas de FPS apresentadas nas seções anteriores, é possível perceber uma pequena vantagem da OpenGL sobre o Directx neste quesito. A Figura 6-12 apresenta este quadro em relação ao método com uso de volumes limites (USE_BOUNDS).

É importante salientar que a biblioteca OpenGL, pela sua portabilidade, é amplamente utilizado pela indústria de desenvolvimento. Para aplicações baseadas no DirectX, haverá necessidade de verificar a compatibilidade com o ambiente de execução a ser escolhido. Também deve ser considerado o fato das classes da J3D, assim como a linguagem Java, serem interpretadas em tempo de execução pela Máquina Virtual Java (JVM - *Java Virtual Machine*). Neste ponto, projetos que utilizam a Microsoft JVM (MSJVM) devem prever sua transição para a plataforma .Net ou concorrente em Java, uma vez que a Microsoft não oferecerá mais suporte à MSJVM a partir de 2008, conforme acordo entre a Sun Microsystems e a própria Microsoft (MICROSOFT CORPORATION, 2004).

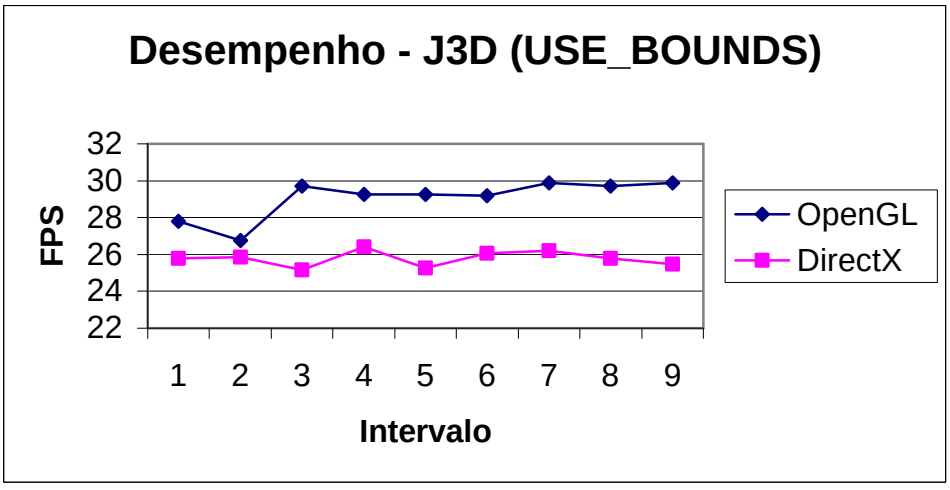


FIGURA 6-12: J3D (USE_BOUNDS) – OPENGL VS DIRECTX.

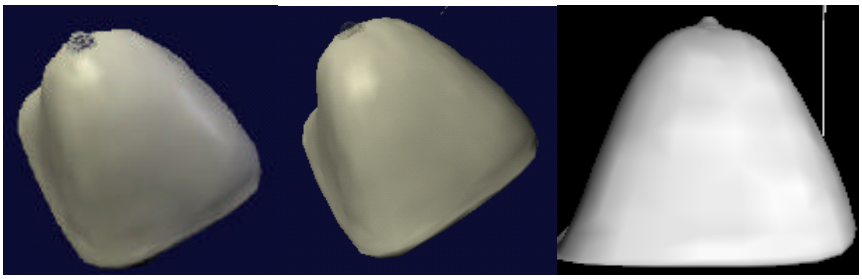
Já entre a J3D e o WTK, é possível fazer uma comparação desde o desenvolvimento até a execução da aplicação. Na etapa de desenvolvimento da aplicação, por exemplo, a grande vantagem do WTK foi a simplicidade na utilização dos métodos e um número considerável de funções já definidas. Na J3D, por sua vez, é necessário informar vários parâmetros para trabalhar com os objetos, desde parâmetros de habilitação para buscar as coordenadas centrais dos objetos até a leitura de dados de suas geometrias, além de que várias implementações ficam por conta do desenvolvedor. Na Figura 6-13, por exemplo, é possível verificar as várias parametrizações que foram informadas para o objeto shapeAgulha, pelo método setCapability, para utilização da geometria do objeto agulha na J3D.

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

	try
	{
		agulha = objFileloader.load("object/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.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;

}
```

FIGURA 6-13: UTILIZAÇÃO DO OBJETO AGULHA NA J3D.

Quanto à utilização dos objetos modelados no 3DStudio, a J3D apresentou melhores resultados quanto à qualidade das imagens geradas a partir da importação dos modelos, não havendo perda significativa da textura original. Como exemplo, a Figura 6-14 mostra o aspecto visual do objeto mama em cada tecnologia.

Em relação à documentação disponível para consultas, apesar da J3D possuir uma versão *on line* de todas as classes e suas sintaxes, os exemplos são poucos e o desenvolvedor deve procurar outras fontes, como fóruns e tutoriais disponíveis na Internet.



a) Modelo Original **b) Na J3D** **c) No WTK**

FIGURA 6-14: QUALIDADE DOS MODELOS GERADOS NA J3D E NO WTK.

Já o WTK traz um manual bem completo, onde todos os métodos disponíveis em sua biblioteca possuem sintaxe e exemplos associados, oferecendo um bom suporte, principalmente para iniciantes. Mas é válido lembrar que o WTK é um software comercial, assim este pode ser um item a considerar no fator benefício em relação ao custo de aquisição.

Quanto ao desempenho, o WTK teve um resultado um pouco melhor em relação a J3D baseada no DirectX. Como sua base é a linguagem C em conjunto com a biblioteca OpenGL, houve pouca superioridade à versão da J3D baseada na OpenGL. Mas é importante salientar que, assim como a linguagem Java, as classes da J3D também são interpretadas em tempo de execução pela JVM. Os resultados podem ser observados na Figura 6-15, onde foram comparados os métodos com uso de volumes limites.

Considerando a empregabilidade de cada tecnologia, isto é, em quanto do mercado dos projetos de RV elas estão presentes, é possível analisar apenas a aplicação na medicina, uma vez que somente esta área foi ponto de pesquisa deste trabalho. Desta maneira, pode-se afirmar pela Tabela 2-1, que o uso destas tecnologias é ainda incipiente. Enquanto nenhum projeto utilizou a J3D como ambiente de desenvolvimento, apenas um, Gorman et al (2000), escolheu o WTK como a tecnologia de *software* para desenvolvimento de sua aplicação. Entretanto, o uso de bibliotecas próprias das tecnologias de *hardware* selecionadas nos projetos, como dispositivos hápticos e estações gráficas, podem ter levado à ampla utilização da linguagem C em conjunto com a biblioteca OpenGL, uma vez que os *drivers* destes equipamentos são usualmente escritos em C ou C++.

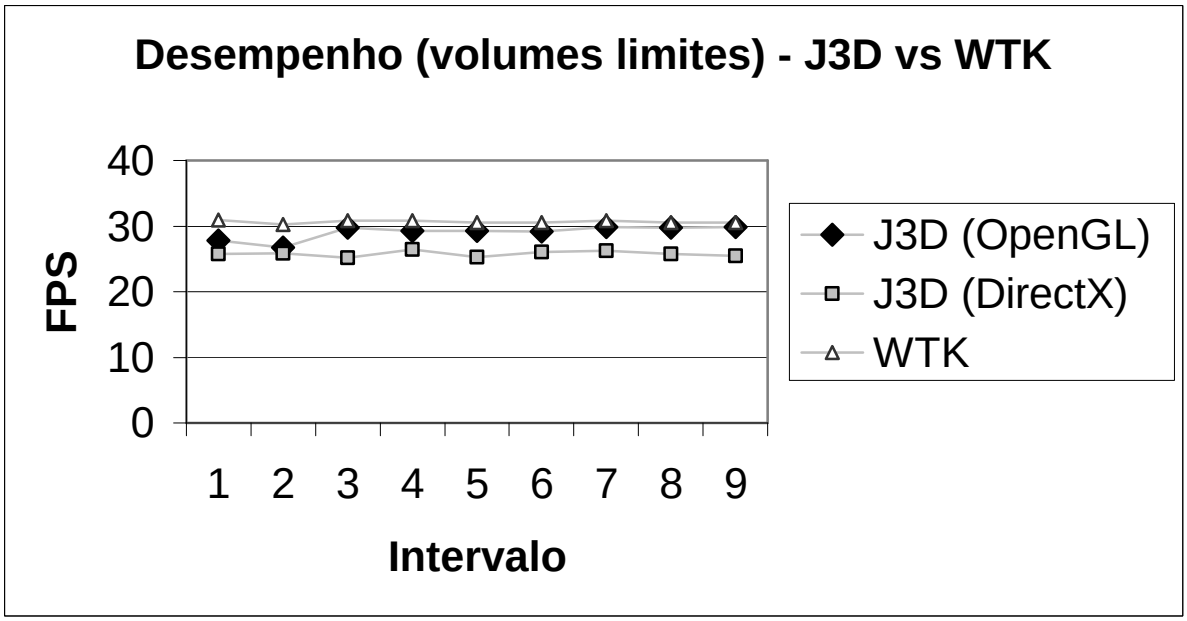


FIGURA 6-15: J3D VS WTK (USO DE VOLUMES LIMITES).

Já em relação aos métodos oferecidos para detecção de colisão, como os métodos baseados no uso de volumes limites são muito imprecisos e os baseados na geometria dos objetos tem alto custo computacional, em ambas as tecnologias há necessidade de refinamento, principalmente na criação de aplicações onde a certeza sobre a intersecção dos

objetos é um fator preponderante, como é o caso de aplicações para treinamento médico, foco deste trabalho.

Por fim, analisando ambas as tecnologias do ponto de vista do desenvolvedor, independente da experiência em determinada linguagem de programação, a utilização do WTK é uma escolha a ser considerada para desenvolvedores iniciantes em aplicações de RV, principalmente se a orientação a objeto for um tema não bem entendido. Além disso, benefícios como documentação e um número considerável de funções já definidas, é um outro fator a ser observado.

No entanto, o custo de aquisição pode ser o maior entrave, principalmente para liberação da aplicação final. Conforme Sense8 (2003), o custo por licença para distribuição comercial do WTK é de $ 1.000,00 e a licença para distribuição comercial ilimitada custa $ 15.000,00.

CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou, além de um estudo sobre os métodos e principais abordagens oferecidas por tecnologias de *software* de RV para detecção de colisão em simuladores de procedimento médico, uma análise sobre os métodos existentes na J3D e no WTK, uma proposta de refinamento com maior precisão e menor custo computacional e, por fim, uma análise comparativa envolvendo todos os métodos implementados nas diferentes tecnologias.

Observou-se que os métodos fornecidos pela J3D e WTK podem não ser suficientes para uma detecção de colisão com precisão e bom desempenho, características que constituem aspectos fundamentais para o tipo de aplicação definido no escopo deste trabalho. O desempenho dos métodos precisos, aqueles que envolvem a geometria dos objetos, é dependente da complexidade destes.

O refinamento dos métodos, por sua vez, com a utilização de um limiar que estabelece uma distância mínima entre dois pontos (distância euclidiana) e a verificação de um ponto sobre a superfície do objeto alvo (equação do plano) para considerar a existência de colisão mostrou-se promissor, mas ainda não satisfatório em algumas situações em relação às necessidades das aplicações de RV para treinamento médico, já que houve momentos em que a intersecção dos objetos foi detectada quando, visivelmente, esta não ocorrera.

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 nas quais o fator precisão não seja crítico.

Quanto à escolha entre a biblioteca J3D e o software WTK para desenvolvimento de uma aplicação médica em RV, fica a cargo do desenvolvedor analisar o custo em relação ao benefício para uma decisão adequada. Documentação e o conjunto de funções existente em cada tecnologia foram pontos importantes no desenvolvimento deste trabalho, além da

necessidade de um bom conhecimento sobre orientação a objetos quando trabalhando com a linguagem Java e a biblioteca J3D.

O WTK oferece um amplo conjunto de funções superior a J3D, além de suporte a uma grande faixa de dispositivos não convencionais, como sensores, *displays* e dispositivos de retorno. Além disso, como a maioria dos *drivers* destes dispositivos são escritos em C ou C++, não é fácil portar uma aplicação J3D para uma nova plataforma.

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:

- Método para determinar limiar automaticamente a partir da implementação de rotinas de animações pré-definidas.
- Método de teste de desempenho sem a intervenção do usuário, também por rotinas de animações pré-definidas.
- 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

AMES, A. L.; NADEAU, D. R.; MORELAND, J. L. **The VRML 2.0 Sourcebook**, 2nd ed. New York: John Wiley & Sons, 1997, 654p.

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.

CÂMARA, G.; DAVIS, C.; MONTEIRO, A. M. **Introdução à Ciência da Geoinformação**. São José dos Campos: INPE, 2004.

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.

COMEAU, C. P.; BRYAN, J. S. Headsight Television System Provides Remote Surveillance. **Electronics**, p. 86-90, nov. 1961.

DIMAIO, S. P.; SALCUDEAN, S. E. Needle Insertion Modelling and Simulation. In: THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, 2002. **Proceedings…** 2002, p. 2098-2105.

DISCREET. Disponível em: <http://www.discreet.com>. Acesso em: 13 março 2004.

FREITAS, C. M.; MANSSOUR, I. H.; NEDEL, L. P.; GAVIÃO, J. K.; PAIM, T. C.; MACIEL, A. Framework para Construção de Pacientes Virtuais: Uma Aplicação em Laparoscopia Virtual. In: SIMPÓSIO DE REALIDADE VIRTUAL, 2003. **Anais...** Ribeirão Preto, 2003, p. 283-294.

GARCIA-ALONSO, A.; SERRANO, N.; FLAQUER, J. Solving the Collision Detection Problem. **IEEE Computer Graphics and Applications**, v.14, n.3, p. 36-43, 1994.

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.

HAND, C. Other Faces of Virtual Reality. In: FIRST INTERNACIONAL CONFERENCE MHVR – LECTURE NOTES IN COMPUTER SCIENCE, 1077, 1994, Moscow. Moscow: Ed. Springer, 1994, p. 107-116.

HUDSON, T. C.; LIN, M. C.; COHEN, J.; GOTTSCHALK, S.; MANOCHA, D. V-COLLIDE: Accelerated Collision Detection for VRML. In: THE SECOND SYMPOSIUM ON VIRTUAL REALITY MODELING LANGUAGE, 1997, Monterey. **Proceedings...** Monterey, 1997, p. 117-123.

IMMERSION MEDICAL. Disponível em: <http://www.immersion.com>. Acesso em: 10 dezembro 2003.

J3D.ORG - THE JAVA 3D COMMUNITY SITE. Disponível em: <http://www.j3d.org>. Acesso em: 03 março 2004.

JACOBSON, L. **Realidade Virtual em Casa**. Rio de Janeiro: Berkeley, 1994, 446p.

JAVA - THE SOURCE FOR JAVA TECHNOLOGY. Disponível em: <http://java.sun.com>. Acesso em: 10 dezembro 2003.

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.

KISMET - 3D SIMULATION *SOFTWARE*. Disponível em: <http://www-kismet.iai.fzk.de>. Acesso em: 20 novembro 2003.

KÜHNAPFEL, U.; ÇAKMAK, H. K.; MAAB H. Endoscopic Surgery Training using Virtual Reality and Deformable Tissue Simulation**. Computers & Graphics**, v. 24, p. 671-682, 2000.

LAPSIM SYSTEM. Disponível em: <http://www.surgical-science.com>. Acesso em: 10 dezembro 2003.

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 SYMPOSIUM ON VIRTUAL REALITY SOFTWARE AND TECHNOLOGY, Hong Kong. **Proceedings…** Hong Kong, 2002, p. 113-120.

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. **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.; 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.; 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.

MANSSOUR, I. H. **Introdução à OpenGL**. Disponível em: <http://www.inf.pucrs.br/~manssour/OpenGL/index.html>. Acesso em: 20 novembro 2003.

MCCARTHY, A. D.; HOLLANDS R. J. Human Factors Related to a Virtual Reality Surgical Simulator: The Sheffield Knee Arthroscopic Training System. In: THE FOURTH UK VR-SIG CONFERENCE, 1997, United Kingdom. **Proceedings…** United Kingdom, 1997.

MICROSOFT CORPORATION. Disponível em: <http://www.microsoft.com>. Acesso em: 15 novembro 2004.

MONTGOMERY, K; HEINRICHS, L.; BRUYNS, C.; WILDERMUTH, S.; HASSER, C.; OZENNE, S.; BAILEY, D. Surgical Simulator for Hysteroscopic: A Case Study of Visualization in Surgical Training. In: IEEE VISUALIZATION CONFERENCE, 12, 2001, San Diego. **Proceedings…** San Diego, 2001.

MYSQL: THE WORLD'S MOST POPULAR OPEN SOURCE DATABASE. Disponível em: <http://www.mysql.com>. Acesso em: 10 março 2004.

NEDEL, L. P. Modelagem e Animação de Superfícies Deformáveis. In: LATIN AMERICAN INFORMATICS CONFERENCE, 18, 1992, Las Palmas. **Proceedings...** Las Palmas, 1992.

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.

OPENGL - OVERVIEW. Disponível em: <http://www.opengl.org>. Acesso em: 10 dezembro 2003.

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

PIMENTEL, K.; TEIXEIRA, K. **Virtual Reality – through the new looking glass**. 2.ed. New York: McGraw-Hill, 1995.

PONAMGI, M.; MANOCHA, D.; LIN, M.C. Incremental Algorithms for Collision Detection Between Solid Models. In: ACM SIGGRAPH SYMPOSIUM ON SOLID MODELING AND APPLICATIONS, 1995, Utah. Utah, 1995.

PONDER, M.; HERBELIN, B.; MOLET, T.; SCHERTENLIEB, S.; ULICNY, B.; PAPAGIANNAKIS, G.; THALMANN, N.M.; THALMANN, D. Immersive VR Decision Training: Telling Interactive Stories Featuring Advanced Virtual Human Simulation Technologies. In: 7TH INTERNATIONAL IMMERSIVE PROJECTION TECHNOLOGIES WORKSHOP AND 9TH EUROGRAPHICS WORKSHOP ON VIRTUAL ENVIRONMENTS, 2003, Zurich. Zurich, 2003.

SATAVA, R. Medicine 2001: The King is Dead. In: VIRTUAL REALITY CONFERENCE, 1994, CA. **Proceedings…** CA, 1994.

SENSABLE TECHNOLOGIES. Disponível em: <http://www.sensable.com>. Acesso em: 20 novembro 2003.

SENSE8 – VIRTUAL REALITY 3D SOFTWARE. Disponível em: <http://sense8.sierraweb.net>. Acesso em: 20 novembro 2003.

SOURINA, O.; SOURIN A.; SEN H. T. Virtual Orthopedic Surgery Training on Personal Computer. **Internacional Journal of Information Technology**, v. 6, n. 1, p. 16-29, 2000.

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.

V-CLIP COLLISION DETECTION LIBRARY. Mitsubishi Electric Research Laboratories. Disponível em: <http://www.merl.com/projects/vclip>. Acesso em: 10 dezembro 2003.

WAGNER, C.; SCHILL, M. A.; MÄNNER, R. Collision Detection and Tissue Modeling in a VR-Simulator for Eye Surgery. In: EUROGRAPHICS WORKSHOP ON VIRTUAL ENVIRONMENTS, 8, 2002, Barcelona. **Proceedings…** Barcelona, 2002.

WEBSTER, R.; HALUCK, R. S.; ZOPPETTI, G.; BENSON, A.; BOYD, J.; CHARLES, N.; REESER, J. SAMPSON, S. A Haptic Surgical Simulator for Laparoscopic Cholecystectomy Using Real-Time Deformable Organs. In: THE IASTED INTERNACIONAL CONFERENCE. BIOMEDICAL ENGINEERING, 2003, Salzburg. **Proceedings…** Salzburg, 2003.