ESCOLA DE ARTES, CIÊNCIAS E HUMANIDADES

EACH – USP

PROGRAMA ENSINAR COM PESQUISA 2009

IMPLEMENTAÇÃO DE REALISMO EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

RELATÓRIO FINAL

Paulo Rodrigues Felisbino

São Paulo – Março/2010

Resumo

O objetivo deste trabalho é aperfeiçoar o realismo do framework *ViMeT* (*Virtual Medical Training*), investigando e implementando ações a respeito de texturas, sombras e outros requisitos necessários para o desenvolvimento, No relatório parcial foi apresentada a revisão bibliográfica incluindo conceitos de Realidade Virtual e trabalhos realizados na área de Medicina; uma introdução sobre a API Java3D e o conceito de grafo de cena; uma introdução ao framework ViMeT e seu diagrama de classes, detalhando as classes que necessitavam de alteração. A fim de contextualizar o projeto, no presente relatório é reapresentada uma introdução com os objetivos e justificativas do projeto. Em seguida, são apresentadas as implementações efetuadas e os resultados obtidos, além da conclusão final obtida com o término do projeto.

Sumário

1. Introdução

A Realidade Virtual é muito empregada no treinamento de profissionais de diversas áreas, Na área médica, os sistemas vêm conquistando seu espaço no tratamento de algumas patologias e simulação de procedimentos.

O *ViMeT* (*Virtual Medical Training*) é um framework em desenvolvimento, voltado para o treinamento médico, mais especificamente, ao treinamento de exames de biópsia. Permite a criação de um ambiente virtual com objetos virtuais representando um órgão humano e um instrumento médico, como uma seringa. As aplicações por ele geradas podem ser controladas utilizando equipamentos convencionais (teclado e mouse) ou equipamentos não convencionais (dispositivo háptico e luva de dados). O presente projeto visa a contribuir com o desenvolvimento do referido framework, conforme apresentado a seguir.

1.1 Objetivos

O principal objetivo deste trabalho é aperfeiçoar o realismo do framework *ViMeT* (*Virtual Medical Training*). Para tanto, está dividido em 4 objetivos específicos:

- Estudar e implementar o uso de texturas e sombras nos objetos 3D que representam órgãos humanos.
- Melhorar o desempenho de um método de deformação já implementado, fazendo com que um objeto flexível que representa um órgão humano volte ao seu estado inicial após a deformação.
- Implementar a característica de relevo no procedimento de deformação, por meio de reposicionamento de vértices do objeto 3D.
- Alterar o framework ViMeT para incluir a finalização dos exames de biópsia, simulando o depósito do material coletado em uma lâmina virtual.

1.2 Justificativa

Os simuladores de RV oferecem inúmeras vantagens se comparados com os métodos tradicionais de ensino e treinamento, principalmente na medicina, como a possibilidade de repetir o procedimento quantas vezes forem necessárias, aumentando assim a experiência do profissional, a minimização da necessidade de se utilizar cadáveres e/ou animais vivos, a possibilidade de visualizar e gravar todos os procedimentos realizados a fim de melhorar as técnicas empregadas ou simplesmente para posterior estudo, a possibilidade de se configurar qualquer situação real, dependendo dos objetivos do docente, entre outras (MONSERRAT et al., 2003).

No entanto, de forma a maximizar o aprendizado, devemos primar pelo realismo desses simuladores. Tendo em vista esse objetivo, este projeto visa contribuir tanto com a área médica quanto com a área de desenvolvimento de sistemas (Sistemas de Informação), uma vez que para realizar este projeto serão necessários conhecimentos técnicos de programação, que podem ser usados inclusive para ensino de engenharia de software, banco de dados, realidade virtual, programação, entre outros.

1.3 Disposição do Trabalho

Este trabalho está dividido da seguinte forma, além desta Introdução:

Seção 2. Desenvolvimento do Projeto: traz as atividades realizadas e o cronograma de execução.

Seção 3. Conclusões: apresenta as conclusões obtidas após o término do projeto.

Seção 4. Referências: traz a relação da leitura base deste projeto.

Apêndice: apresenta os códigos fontes das classes modificadas e criadas.

2. Desenvolvimento do Projeto

Nesta seção são apresentadas as implementações efetuadas e os resultados obtidos (divididos em três partes: Textura, Sombra e Lâmina), juntamente com o cronograma de execução deste projeto.

Após estudo das classes que compõem o *ViMeT*, ficou decidido que para incrementar realismo, seria necessário modificar a classe *Environment*, responsável pela criação do ambiente 3D. O diagrama de classes do *ViMeT* pode ser visto na Figura 1. Nele, é possível observar todas as classes que compõem o framework, assim como os relacionamentos existentes com a classe *Environment*. A Figura 2 mostra a representação da classe *Environment*, que contém sete métodos, a saber:

- *Environment*: construtor responsável pela criação do AV em si (nó *Locale*, etc).
- *setEyeOffset*: responsável por setar a distância da “visão” utilizada na estereoscopia.
- *buildViewBranch*: responsável pela criação do nó que estabelece a posição e visualização do ambiente.
- *add*: responsável por adicionar os objetos modelados ao Universo Virtual.
- *add2*: responsável por adicionar os objetos modelados ao Universo Virtual.
- *add3*: responsável por adicionar os objetos modelados ao Universo Virtual.
- *addGrafo*: adiciona o grafo de cena no nó *Locale*.

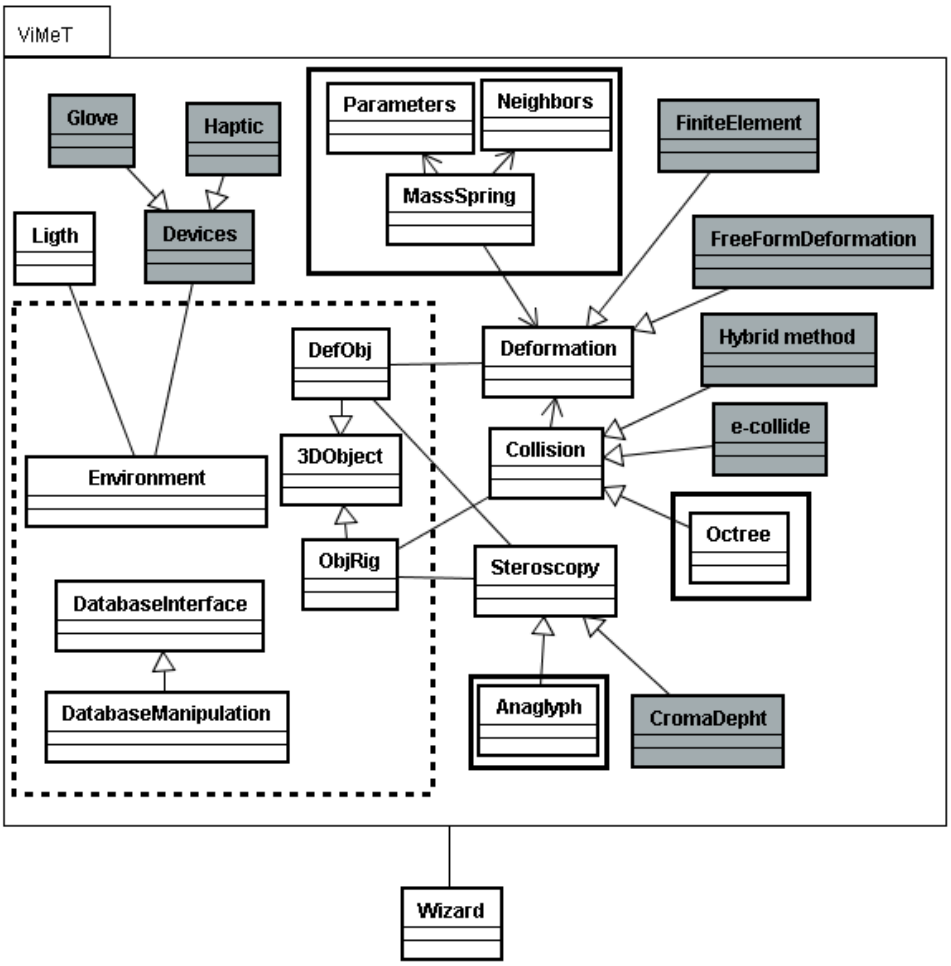

Figura 1 – Diagrama de classes do *ViMeT*

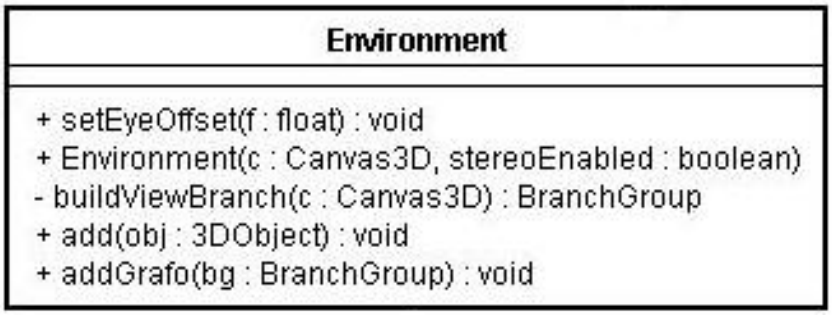

Figura 2 – Diagrama de classes da classe *Environment*.

Durante a instalação do *ViMeT*, foi utilizado o *Manual de Instanciação do Framework ViMeT* (OLIVEIRA, 2007). Primeiramente, foi instalado o banco de dados Derby. Para tanto, foi necessário configurar as variáveis de ambiente DERBY_HOME, CLASSPATH e Path. Após a instalação do banco de dados,

foi executada a classe *Wizard,* responsável pela interface gráfica (Figura 3) e instanciação do *ViMeT*.

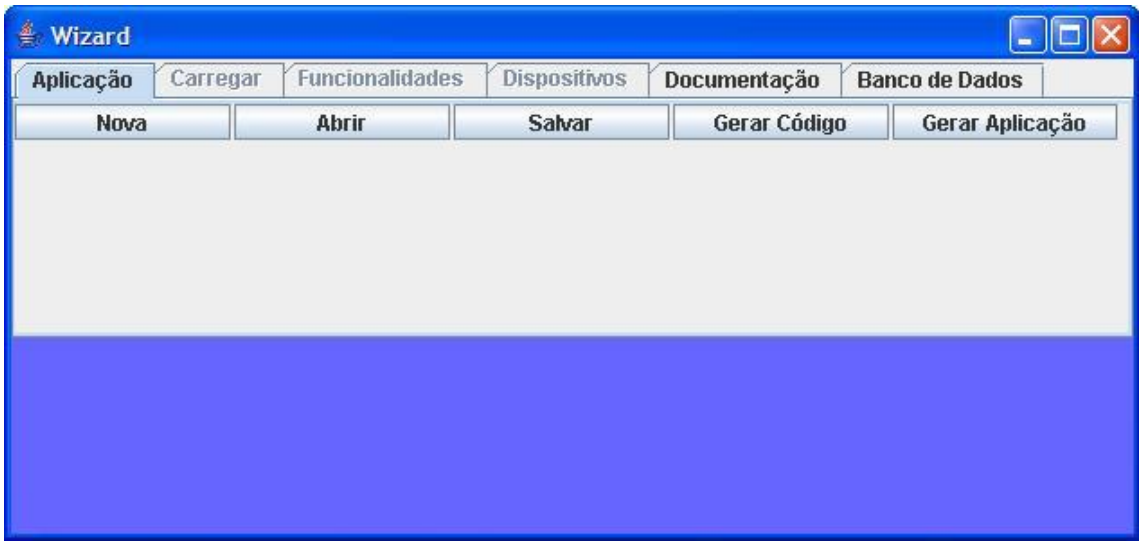

Figura 3 – Interface da *Wizard*.

Em seguida, decidiu-se retomar a pesquisa bibliográfica, a fim de verificar como poderia ser implementado realismo nos objetos 3D. Dois aspectos foram selecionados para abordagem inicial: sombra e textura.

2.1 Textura

Conforme foi citado anteriormente, a aplicação de textura é muito importante quando falamos de RV, principalmente em aplicações na área médica. Quando falamos sobre o *Virtual Medical Training* (*ViMeT*), a situação não é diferente.

No teste de textura, foi utilizada uma imagem no formato "*jpeg*", com dimensões 128 x 128 pixels. Essa imagem foi obtida da seguinte forma: primeiramente, foi tirada uma foto bem próxima da pele. Logo após, a foto foi recortada e ampliada (Figura 4).

Figura 4 – Foto de pele humana ampliada

Originalmente, a mama era apresentada no modo *wireframe* (linhas), sem nenhuma textura (Figura 5).

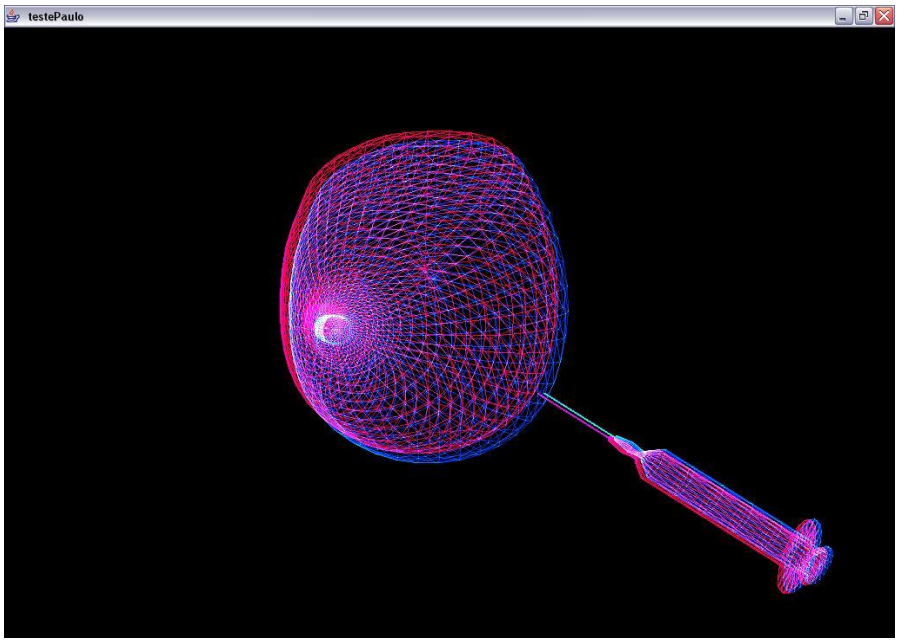

Figura 5 – Execução do *ViMeT* - *wireframe*

Para acrescentar textura na mama, foi necessária uma pequena modificação em outras duas classes do *ViMeT* (*ObjDef* e *Object3D*), além da classe que já estava prevista (*Environment*). A alteração dessas duas classes surgiu após a constatação de um problema: quando a textura foi colocada com êxito, ela era aplicada em todos os objetos do AV (o órgão humano e a seringa).

Na classe *Object3D* foi criada uma variável booleana chamada *ehOrgao* (que indica se o objeto que está sendo carregado é um órgão ou uma seringa) e um método chamado *podeSerOrgao()* (que retorna o valor da variável booleana).(Figura 6)

```
public boolean ehOrgao = false;
public boolean podeSerOrgao() {
return ehOrgao;
}
```

Figura 6 – Trecho de código da classe *Object3D*

A classe *ObjDef* (subclasse de *Object3D*) possui os métodos de carregamento e funcionalidades do objeto modelado que simula um órgão (objeto deformável). Um dos construtores dessa classe (*ObjDef(String nome, int modo, int modoObjectFile)*), carrega o órgão. Se o órgão for carregado com sucesso, a variável *ehOrgao* se torna *true* (Figura 7).

```
public ObjDef(String nome, int modo, int modoObjectFile) {
  Scene mama = null;
  objFileloader = new ViMeT.DefApliMedLoader.ObjectFile(modoObjectFile);
  try {
    mama = objFileloader.load(nome);
    ehOrgao = true;
  } catch (Exception e) {
    mama = null;
    System.err.println(e);
  }
}
```

Figura 7 – Trecho de código da classe *ObjDef*.

Por último, na classe *Environment*, no método *add(Object obj)*, foi incluído o IF abaixo (Figura 8).

```
if(obj.podeSerOrgao() == true){
  TextureLoader loader = new TextureLoader("peleTeste.jpg", null);
  ImageComponent2D image = loader.getImage();
  Texture2D texture = new Texture2D (Texture.BASE_LEVEL,Texture.RGBA,image.getWidth(),image.getHeight());
  texture.setImage(0, image);
  texture.setEnable(true);
  ap.setTexture(texture);
}
```

Figura 8 – Trecho de código da classe *Environment*.

Se o órgão humano for carregado com sucesso (*ehOrgao* = *true*), a textura é aplicada, caso contrário, não é aplicada nenhuma textura (válido quando a Wizard carrega a seringa). O resultado do órgão com a textura já aplicada pode ser visto na Figura 9.

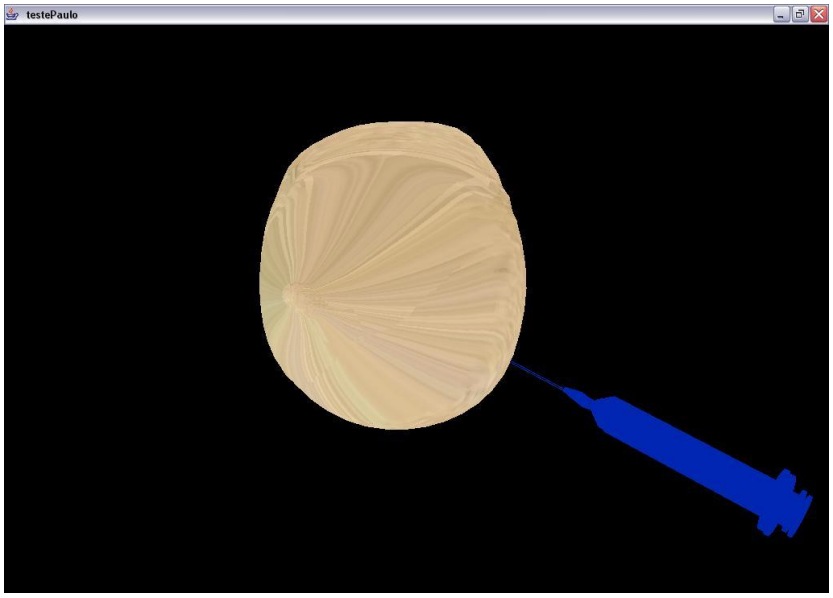

Figura 9 – Execução do ViMeT – textura

É possível observar que o resultado desta textura não atingiu o nível de realismo almejado, porém um passo foi dado nesse sentido.

É importante ressaltar que o *ViMeT* pode gerar aplicações com vários outros órgãos além da mama apresentada nesse projeto. A Figura 10 exemplifica o *ViMeT* rodando um glúteo (modo *wireframe*). A Figura 11 mostra um comparativo desse mesmo glúteo no modo *wireframe* e o modo com a textura citada anteriormente.

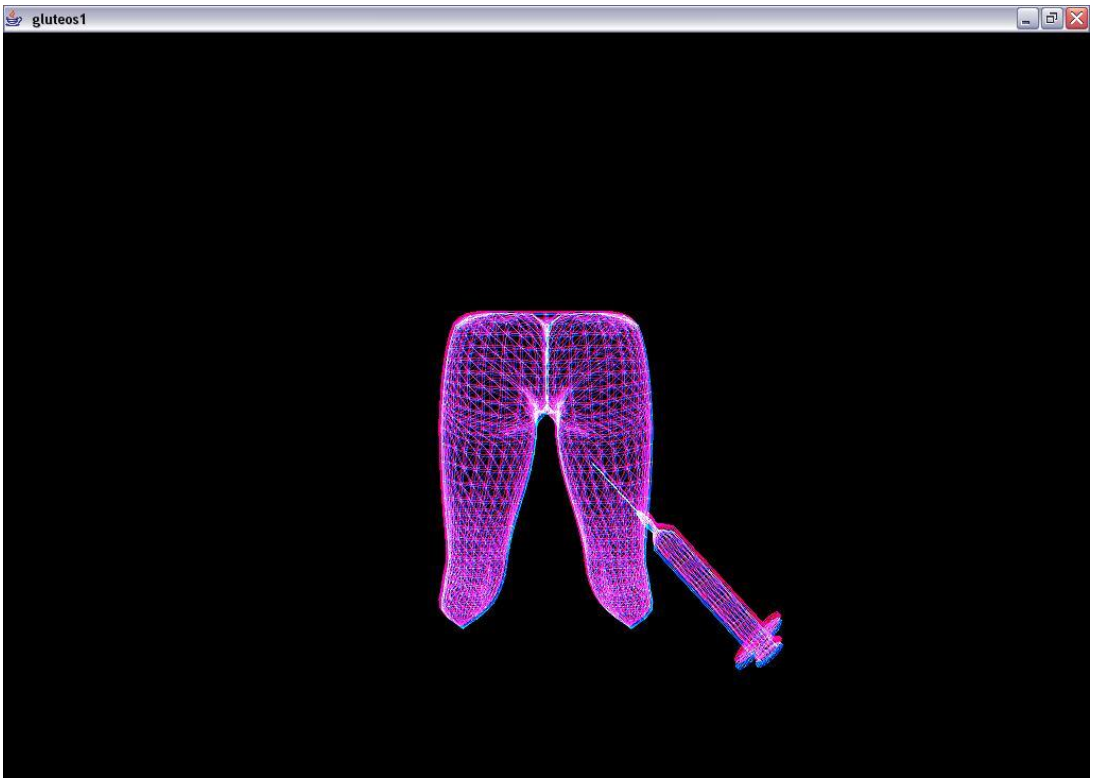

Figura 10 – *ViMeT* – glúteos (*wireframe*)

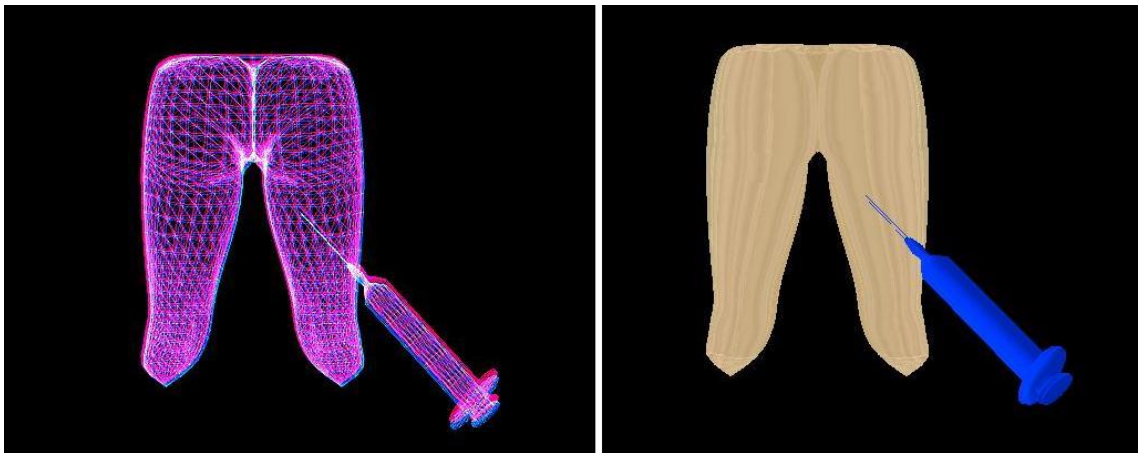

Figura 11 – Comparativo *wireframe*/textura

A seguir, será apresentado o trabalho com sombras e os resultados no aumento do realismo.

2.2 Sombra

O fator sombra também é muito importante em uma aplicação de RV, pois uma de suas funções é propiciar a sensação de profundidade.

Para se obter êxito com o efeito de sombra, pode-se trabalhar com a iluminação do ambiente. A API Java3D disponibiliza 4 tipos de iluminação (Sun, 2009):

- *AmbientLight*: os raios de luz apontam em todas as direções, inundando o ambiente e iluminando todas as formas equilibradamente. (Figura 12)
- *DirectionalLight*: os raios de luz são paralelos e apontam em uma única direção. (Figura 14)
- *PointLight*: os raios de luz são emitidos radialmente a partir de um ponto, em todas as direções. (Figura 16)
- *SpotLight*: os raios de luz são emitidos radialmente a partir de um ponto, na forma de um “cone”. (Figura 18)

A seguir, os resultados obtidos em cada um dos testes:

Teste 1: *AmbientLight*

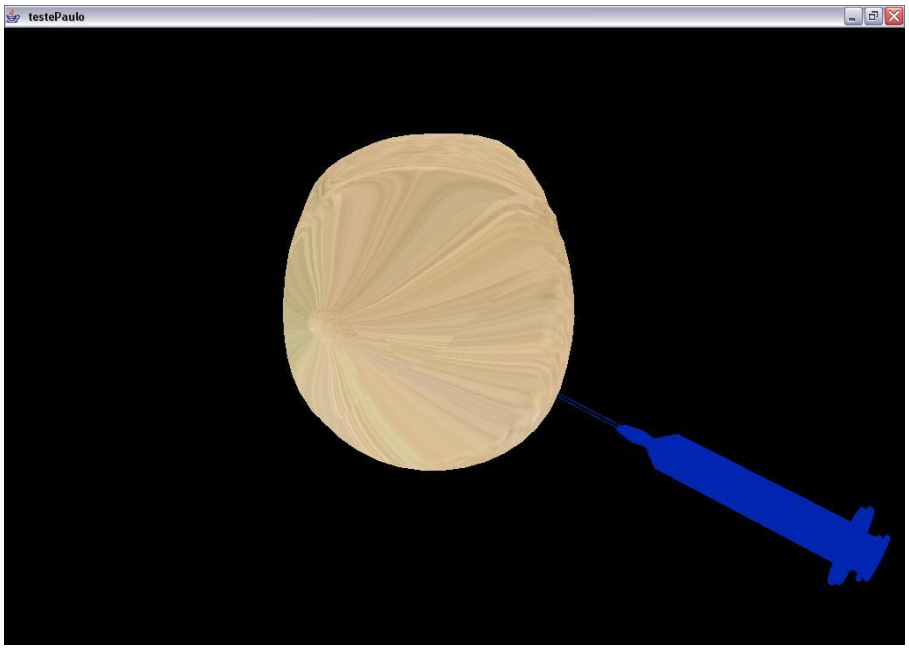

Figura 12 – Execução do *ViMeT – AmbientLight*

Não há muito o que comentar sobre esse teste, uma vez que o *AmbientLight* é a iluminação padrão do *ViMeT*. Podemos notar mais facilmente o efeito deste e os próximos testes se observarmos a seringa. A Figura 13 apresenta o código utilizado.

```
light = new BranchGroup();
BoundingSphere bounds1 = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
Color3f ambientColor = new Color3f(1.0f, 1.0f, 1.0f);
ambientLightNode = new AmbientLight(ambientColor);
ambientLightNode.setInfluencingBounds(bounds1);
light.addChild(ambientLightNode);
myLocale.addBranchGraph(light);
myLocale.addBranchGraph(light);
```

Figura 13 – Trecho de código do *AmbientLight*

Teste 2: *DirectionalLight*

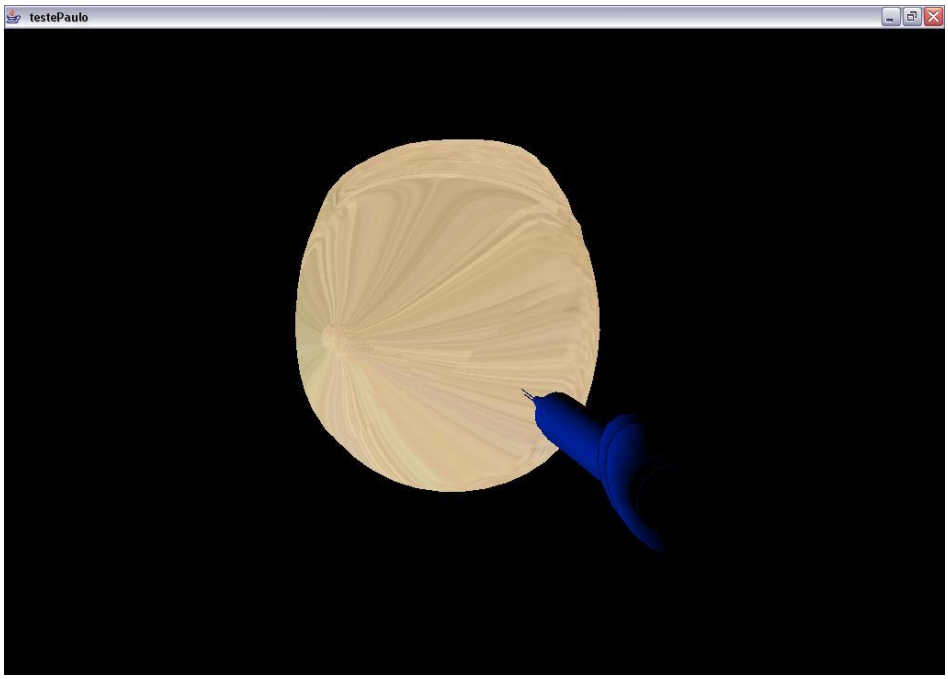

Figura 14 – Execução do *ViMeT* – *DirectionalLight*

Mais uma vez, pode-se observar na seringa o efeito da sombra (o raio de luz segue da esquerda para a direita), embora ainda não seja perceptível nenhuma mudança na mama. A Figura 15 apresenta o código utilizado.

```
light = new BranchGroup();
BoundingSphere bounds1 = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
Color3f light1Color = new Color3f(.9f, 0.9f, 0.9f);
Vector3f light1Direction  = new Vector3f(2.0f, 0f, 0f);
light1 = new DirectionalLight(light1Color, light1Direction);
light1.setInfluencingBounds(bounds1);
light.addChild(light1);
myLocale.addBranchGraph(light);
```

Figura 15 – Trecho de código do *DirectionalLight*.

Teste 3: *PointLight*

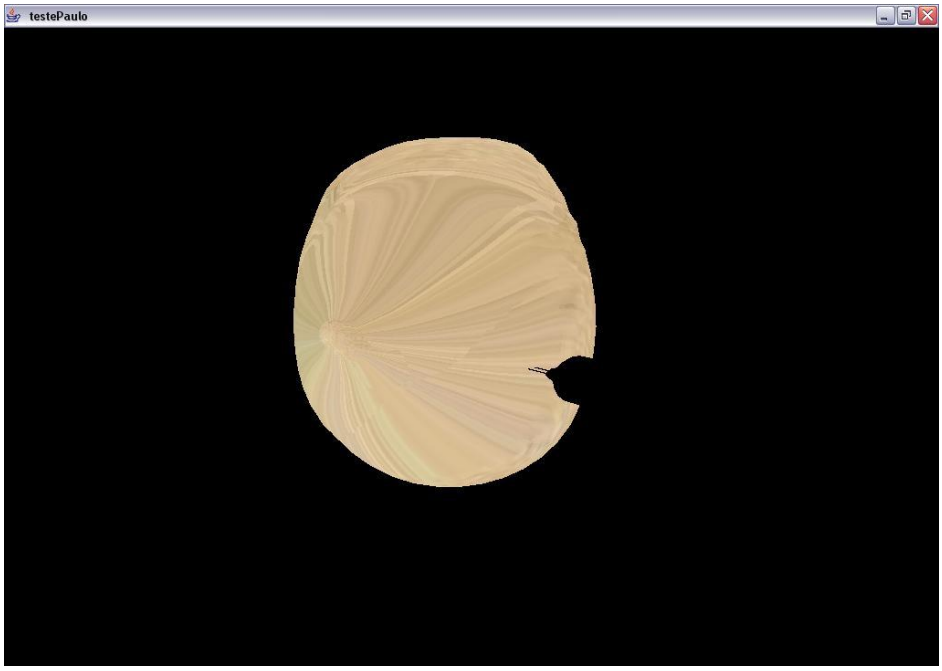

Figura 16 – Execução do *ViMeT – PointLight*

Talvez um dos piores resultados, não importando os valores utilizados, sempre tem-se a impressão de que a mama está irradiando luz ao redor. Notamos que a seringa fica encoberta pelas sombras em quase todas as posições. A Figura 17 apresenta o código utilizado.

```
light = new BranchGroup();
BoundingSphere boundsl = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
Color3f light1Color = new Color3f(.9f, 0.9f, 0.9f);
light1 = new PointLight( );
light1.setEnable(true);
light1.setColor(light1Color);
light1.setPosition(new Point3f(5f, 0f, 0f));
light1.setAttenuation(new Point3f(5f,2f,0.0f));
myLocale.addBranchGraph(light);
```

Figura 17 – Trecho de código do PointLight

Teste 4: *SpotLight*

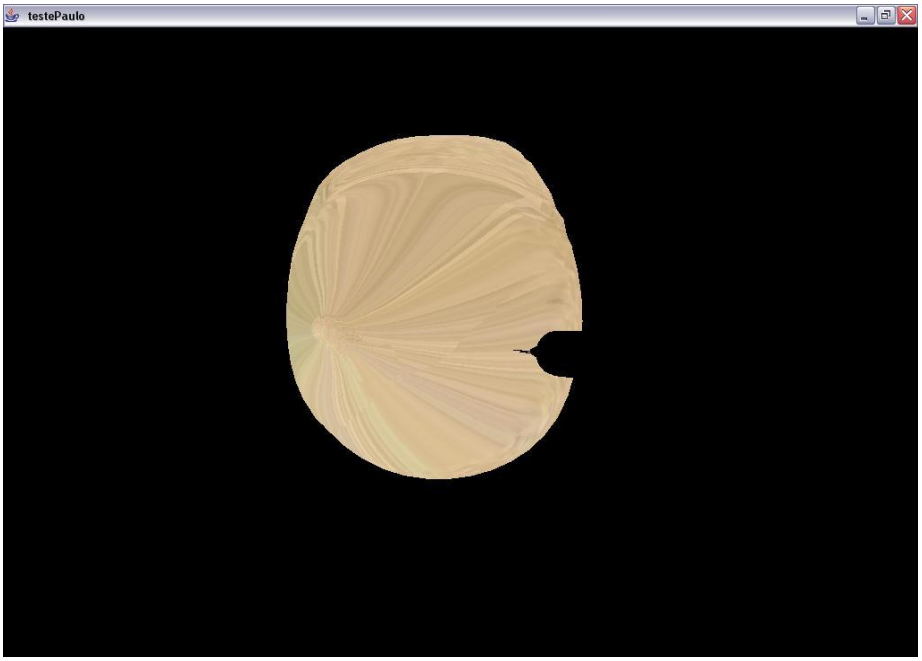

Figura 18 – Execução do *ViMeT* – *SpotLight*

Semelhante ao *PointLight*, a impressão de que a mama é o centro da luz continua. Juntamente com o teste 3, este foi um dos piores resultados. A Figura 19 apresenta o código utilizado.

```
light = new BranchGroup();
BoundingSphere boundsl = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);
Color3f lightlColor = new Color3f(.9f, 0.9f, 0.9f);
Vector3f lightlDirection  = new Vector3f(4.0f, -7.0f, -12.0f);
lightl = new SpotLight();
lightl.setEnable(true);
lightl.setColor(lightlColor);
lightl.setPosition(new Point3f(0.0f,1.0f,0.0f));
lightl.setAttenuation(new Point3f(1.0f,0.0f,0.0f));
lightl.setDirection(lightlDirection);
lightl.setSpreadAngle(0.785f); // 45 degrees
lightl.setConcentration(3.0f); // Unfocused
myLocale.addBranchGraph(light);
```

Figura 19 – Trecho de código do *SpotLight*.

Ao final dos 4 testes, podemos notar que a mama não mostrou alterações significativas de sombra (como foi proposto na seção 1.1 Objetivos), em nenhum dos testes. Porém, é notável as mudanças ocorridas na seringa, destacando-se o 2º teste, com o DirectionalLight.

O código fonte das classes *ObjDef*,*Object3D* e *Environment* podem ser encontrados nos apêndices A, B e C, respectivamente.

2.3 Lâmina

Conforme apresentado na seção 1.1 Objetivos, um dos objetivos propostos foi alterar o framework *ViMeT* para incluir a finalização dos exames de biópsia, simulando o depósito do material coletado em uma lâmina virtual. Infelizmente não foi possível concluir totalmente esse objetivo (ver seção 2.4 Cronograma). No entanto, apresentamos aqui a criação da lâmina e sua inserção no *ViMeT*.

O objeto que representa a lâmina foi criado usando uma das formas geométricas já presentes na API Java3D, o Box. Foram realizados dois testes para a criação da lâmina, um com textura e o outro com atributos de transparência.

A textura aplicada na lâmina é semelhante à textura aplicada na mama: uma imagem no formato “*jpeg*”, com dimensões de 128x128 pixels, obtida a partir de uma fotografia de uma lâmina de vidro real, posteriormente ampliada (Figura 20).

Figura 20 – Foto de uma lâmina de vidro ampliada.

A textura foi aplicada através do código da Figura 21.

```
Appearance vidro = new Appearance();
TextureLoader loader = new TextureLoader("lamina.jpg", null);
ImageComponent2D image = loader.getImage();
Texture2D texture = new Texture2D (Texture.BASE_LEVEL,Texture.RGBA,image.getWidth(),image.getHeight());
texture.setImage(0, image);
texture.setEnable(true);
vidro.setTexture(texture);
```

Figura 21 – Trecho de código da textura na lâmina.

A Figura 22 exemplifica o resultado da lâmina com textura. A Figura 23 mostra a mesma lâmina inserida no *ViMeT*, juntamente com a mama e a seringa.

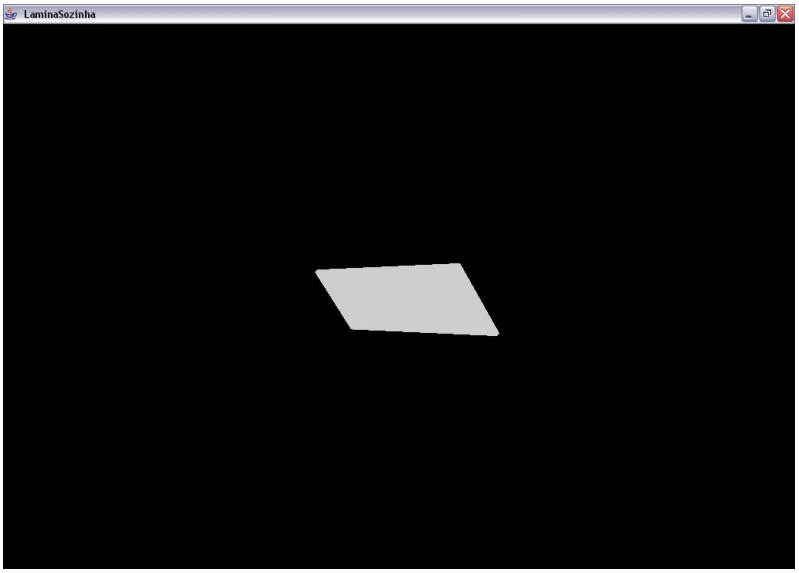

Figura 22 – Lâmina com textura.

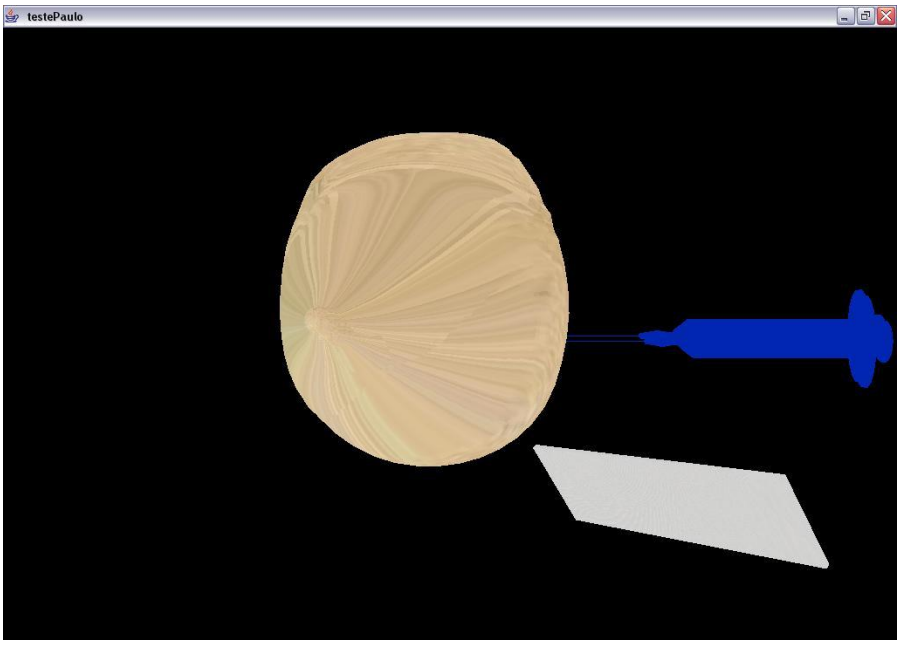

Figura 23 – *ViMeT* com lâmina.

Como podemos observar, a aplicação da textura teve um efeito bom, porém não atingiu o nível de realismo desejado.

O segundo teste, sem textura, e com transparência (Figura 24), foi obtido com a inclusão do código da Figura 25. Na Figura 26, vemos a lâmina transparente inserida no *ViMeT*.

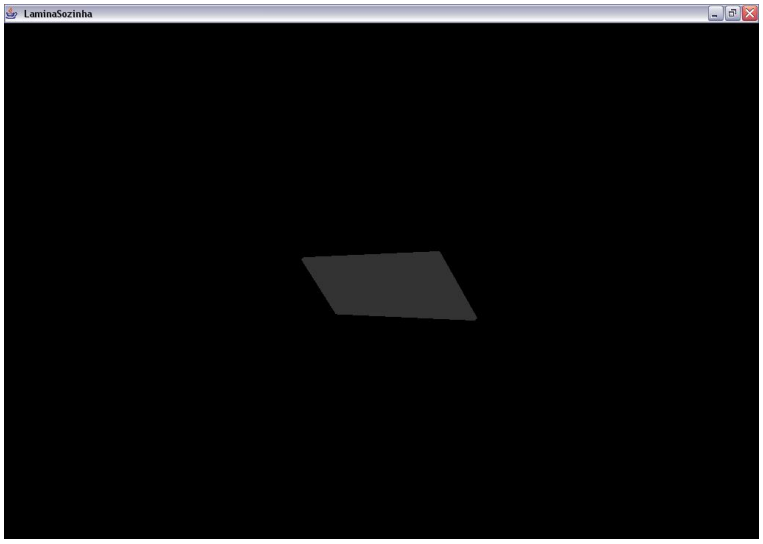

Figura 24 – Lâmina Transparente.

```
Appearance vidro = new Appearance();
TransparencyAttributes t = new TransparencyAttributes(TransparencyAttributes.BLENDED,
                                                      0.8f,
                                                      TransparencyAttributes.BLEND_SRC_ALPHA,
                                                      TransparencyAttributes.BLEND_ONE);
vidro.setTransparencyAttributes(t);
```

Figura 25 – Trecho de código da transparência.

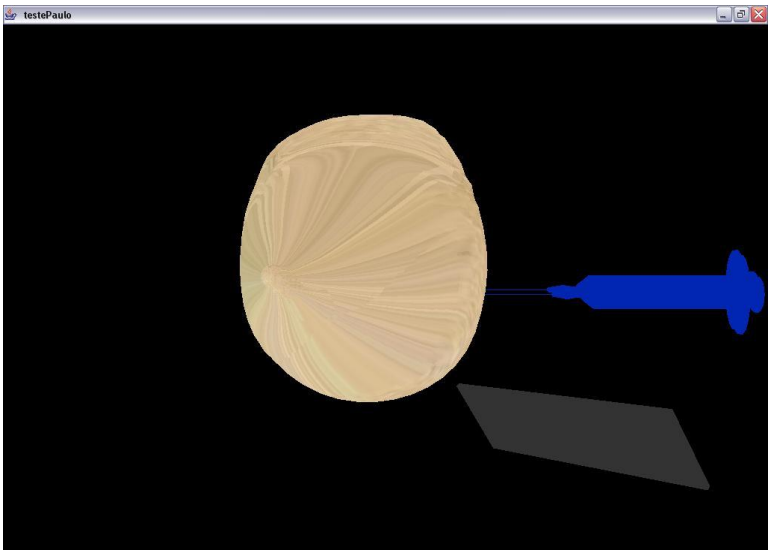

Figura 26 – *ViMeT* com lâmina.

Como podemos observar, a transparência obteve um nível de realismo maior do que a aplicação de textura, embora possa ainda ser melhorado.

O código fonte da classe *Lamina* e uma variação desta criada para gerar as Figuras 22 e 24, podem ser encontrados nos Apêndices D e E, respectivamente.

2.4 Cronograma

A Figura 27 representa o cronograma de execução do projeto, dividido em 8 etapas distintas.

Atividades	Meses											
	1	2	3	4	5	6	7	8	9	10	11	12
1. estudo do framework ViMeT												
2. revisão bibliográfica												
3. implementação de realismo dos objetos 3D												
4. implementação de parâmetros físicos dos tecidos humanos considerados												
5. elaboração do relatório parcial												
6. implementação da finalização do exame de biópsia												
7. execução de testes												
8. elaboração e confecção do relatório final												

	Etapa cumprida
	Etapa parcialmente cumprida
	Etapa não cumprida

Figura 27 – Cronograma de execução do projeto

Como pôde ser observado na Figura 27, algumas etapas que deveriam ter sido concluídas não o foram. Isso se deve a alguns fatos, a saber:

1. Foi refeito o site do grupo de pesquisa *LApiS* (*Laboratório de Aplicação de Informática em Saúde*), onde estão armazenados dados sobre outros projetos conduzidos pela Prof. Dra. Fátima L. S. Nunes Marques. O site foi refeito para que os resultados do Ensinar Com Pesquisa sejam disponibilizados como material didático, conforme previsto nos objetivos do Programa. Desta forma, um tempo considerável do cronograma foi gasto nessa tarefa. Posteriormente este projeto será incluso no *website*. Para maiores informações, acesse <http://each.uspnet.usp.br/lapis/ >. (Figura 28)

2. No início do projeto, não tínhamos um laboratório montado ainda e, portanto, nos dedicamos mais à pesquisa bibliográfica.

3. A etapa 6 não pôde ser concluída uma vez que o tempo se mostrou insuficiente. Os resultados obtidos (inclusão da lâmina no *ViMeT* e os

testes de textura e transparência) foram apresentados na seção 2.3 Lâmina.

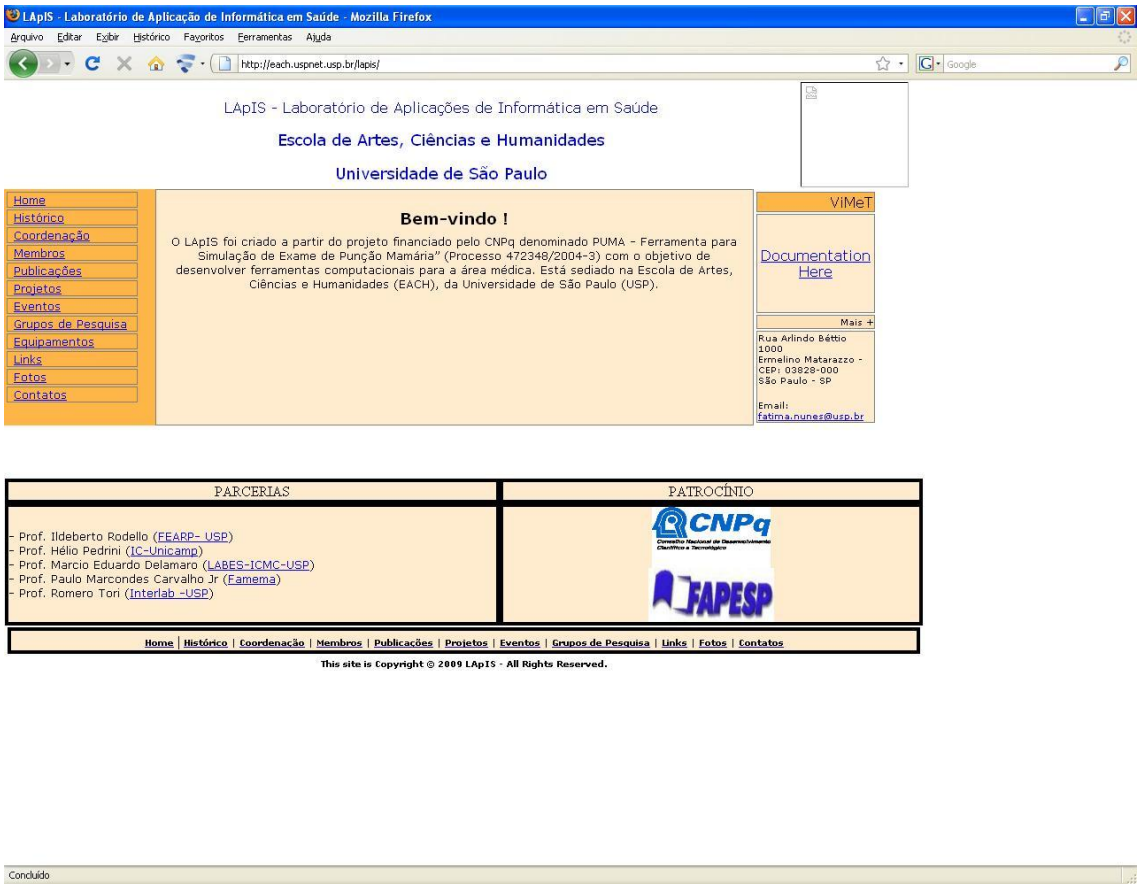

Figura 28 – Página inicial do website do LApiS.

3. Conclusões

Após o término do projeto, ficou claro que a Realidade Virtual tem muito a oferecer à sociedade em geral, seja na educação, através do ensino e treinamento de profissionais de diversas áreas, ou como forma de tratamento para diversas patologias, como o medo de altura, ou o medo de andar de avião.

Podemos enfatizar a importância do *ViMeT* como meio educador da área médica, assim como sua capacidade de proporcionar um treinamento mais eficaz ao usuário. Nota-se também, que o fator realismo utilizado ainda deixa a desejar, porém é preciso trabalhar este aspecto o melhor possível, visando aumentar a qualidade do ensino por ele proporcionado e contribuir ainda mais com a sociedade de um modo geral.

Com relação à aprendizagem proporcionada, é necessário ressaltar sua importância, uma vez que ao se pesquisar sobre determinado assunto, acontece um aprofundamento no tema e assimilação de conceitos a ele relacionados. No caso desse projeto, foi pesquisado a fundo tanto Realidade Virtual - que abrange não só a Medicina (através do desenvolvimento de simuladores cirúrgicos ou para tratamento de fobias), mas muitas outras áreas, como por exemplo, a Arquitetura - quanto à linguagem Java, uma das linguagens fundamentais no curso de Sistemas de Informação. Vários conhecimentos foram adquiridos durante o estudo sobre Java3D, principalmente no que diz respeito à criação da classe Lâmina, que permitiu serem realizados vários testes, pesquisas e, por consequência, proporcionou um grande ganho no aprendizado, enfatizando uma das propostas do Ensinar Com Pesquisa: o desenvolvimento do conhecimento no campo do ensino de graduação. Além disso, foi adquirido um conhecimento um pouco mais amplo sobre o *prompt* de comando, durante a instalação do banco de dados Derby, utilizado pelo *ViMeT*.

4. Referências

BORSHUKOV, G.; LEWIS, J. P. **Realistic human face rendering for "The Matrix Reloaded"**. In: *ACM SIGGRAPH 2003 Sketches &Amp; Applications* (San Diego, California, July 27 - 31, 2003). SIGGRAPH '03. ACM, New York, NY, 1-1. Disponível em: < http://doi.acm.org/10.1145/965400.965470 >. Acesso em: 20 junho 2009

CORRÊA, C. G. **Implementação e Avaliação de Interação em um Framework para Treinamento Médico**. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília. Fundação de Ensino Eurípides Soares da Rocha, Marília, 2008.

GOVINDARAJU, K. N. et al. **Interactive Shadow Generation in Complex Environments**. 2003. Disponível em: <http://portal.acm.org/citation.cfm?id=1201775.882299&coll=Portal&dl=GUIDE&CFID=47461996&CFTOKEN=51403354 >. Acesso em: 01 junho 2009

GOH, K. Y. C.. **Virtual Reality Applications in Neurosurgery**. In: Engineering in Medicine and Biology Annual Conference, 27., 2005, Shanghai. **Proceedings…** p. 4171-4173. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1615383>.Acesso em: 26 julho 2009

LONDON, J.; GEHRINGER M. **Odisséia Digital**. São Paulo: Abril, 2001

LEEB, V. et al., **Interactive Texturing by Polyhedron Decomposition**, 2001. In: Proceedings of the Virtual Reality 2001 Conference (VRí01). Disponível em: < http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00913783 >.Disponível em: 10 abril 2009

MONTERO, E. F. de S.; ZANCHET, D. J. **Realidade virtual e a medicina**. Acta Cir. Bras., São Paulo, v. 18, n. 5, Outubro. 2003. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0102-86502003000500017&lng=en&nrm=iso .Acesso em: 21 jul. 2009.

MONSERRAT, C.; ALCAÑIZ, M.; ULLRICH, M.; POZA, JL.; JUAN, MC.; GRAU, V.**Simulador para el entrenamiento en cirugías avanzadas**.Actas XII congreso internacional de ingeniería gráfica (ISBN: 84-8448-008-9). Disponível em: < http://www.dsic.upv.es/~cmonserr/ Articulos/AA028.pdf > . Acesso em: 08 julho 2009.

MONTEIRO, B. **Videogame Medicinal**. São Paulo: Globo, 2008. Disponível em:< http://revistaepoca.globo.com/Revista/Epoca/0,,EDG66871-6014,00-VIDEOGAME+MEDICINAL.html >. Acesso em: 15 junho 2009

NUNES, F. L. S.; OLIVEIRA, A. C. M. T. G. ; ROSSATO, D. J.; MACHADO, M. I. C. **ViMeTWizard: Uma ferramenta para instanciação de um framework de Realidade Virtual para treinamento médico**. In: XXXIII Conferencia Latinoamericana de Informática, 2007, San José. Proceedings of XXXIII Conferencia Latinoamericana de Informática, 2007. v. 1. p. 1-8.

OLIVEIRA, A. C. M. T. G.; NUNES, F. L. S. **Building a Virtual Medical Training (ViMeT) open source framework**. Journal of Digital Imaging, 2009.

OLIVEIRA, A. C. M. T. G. **ViMeT – Projeto e Implementação de um framework para aplicações de treinamento médico usando realidade virtual**. Dissertação (Mestrado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2007.

OLIVEIRA, A. C. M. T. G. **Cookbook Manual de Instanciação do Framework ViMeT**., 2007.

SANNIER, G.; MAGNENAT THALMANN, N.. **A user-friendly texture-fitting methodology for virtual humans**. 1997. Disponível em: <http://portal.acm.org/citation.cfm?id=1201775.882299&coll=Portal&dl=GUIDE&CFID=47461996&CFTOKEN=51403354>. Acesso em: 27 maio 2009

SOWIZRAL, H. et al., **The Java 3D API Specification**, Second Edition, 2000.

SUDARSKY, S. **Generating Dynamic Shadows for Virtual Reality Applications**, 2001. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=942116>. Acesso em: 11 junho 2009

SUN. Disponível em: < http://java.sun.com/javase/technologies/desktop/java3d/ >. Acesso em: 19 agosto 2009

VRML. **The Virtual Reality Modeling Language**, International standard-ISO/IEC/ 14772- 1 : 1997

YESIL, M.S.; GUDUKBAY, U. **Realistic Rendering and Animation of a Multi-Layered Human Body Model**, *Information Visualization, 2006. IV 2006. Tenth International Conference on* , vol., no., pp.785-790, 5-7 Julho 2006. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1648349&isnumber=34559>. Acesso em: 22 abril 2009

Apêndice

Esta seção apresenta os códigos fontes das classes que foram modificadas e/ou criadas. Um total de 5 classes são aqui apresentadas, numeradas de A até E.

Os apêndices A, B e C mostram o código de classes que já existiam no *ViMeT,* que sofreram algumas alterações.

O apêndice D mostra o código da classe *Lamina*, que foi criada e inserida no *ViMeT*.

O apêndice E mostra uma variação da classe *Lamina*, feita especialmente para a criação das Figuras 22 e 24.

Apêndice A – Código fonte da classe ObjDef.java

```
package ViMeT;
import com.sun.j3d.loaders.objectfile.ObjectFile;
import javax.swing.*;
import java.awt.GraphicsConfiguration;
import java.awt.*;
import java.awt.event.*;
import javax.media.j3d.*;
import javax.media.j3d.Node;
import javax.vecmath.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.vp.*;
import com.sun.j3d.utils.behaviors.*;
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.event.*;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.picking.PickTool;
import javax.swing.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import java.awt.TextArea;
import com.sun.j3d.loaders.Scene;

/**
* Possui os métodos de carregamento,
* funcionalidades do objeto modelado que simula um órgão (objeto deformável)
* @version 15.1 - fev 2007
* @author Ana Cláudia M. T. G. de Oliveira
*/
public class ObjDef extends Object3D{

 Collision cd;
 Deformation def;
 Shape3D shapeMama;
 ViMeT.DefApliMedLoader.ObjectFile objFileloader;
 Shape3D shape2;

/**
*
* @param nome caminho do obejto que simula um órgão humano (objeto deformável).
* @param modo indica quais funcionalidades o objeto possui.
*/
 public ObjDef(String nome, int modo) {
  this(nome,modo,0);
 }
/**
*
* @param nome caminho do obejto que simula um órgão humano (objeto deformável).
```

```
* @param modo indica quais funcionalidades o objeto possui.
* @param modoObjectFile indica qual é o modo de carregamento que será utilizado
pela ObjectFile.
*/
 public ObjDef(String nome, int modo, int modoObjectFile) {
  Scene mama = null;
  objFileloader = new ViMeT.DefApliMedLoader.ObjectFile(modoObjectFile);

  try {
   mama = objFileloader.load(nome);
   ehOrgao = true;
  } catch (Exception e) {
   mama = null;
   System.err.println(e);
  }

  shapeMama = (Shape3D) objFileloader;

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

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

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

  if ((modo&STEREOSCOPY)!=0){
   shape2 = new Shape3D(shapeMama.getGeometry());
   shape2.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
   shape2.setCapability(Shape3D.ALLOW_GEOMETRY_WRITE);
  }

  if ((modo&DEFORMATION)!=0){
   if ((modo&STEREOSCOPY)!=0) {
    def = new MassSpring(objFileloader,shape2);
   }
   else{
    def = new MassSpring(objFileloader);
   }
  }

  if (((modo&0x07)==COLLISION)||((modo&0x07)==OCTREE)){
   BoundingBox bounds = new BoundingBox();
   cd = new Octree(shapeMama);
   cd.setSchedulingBounds(bounds);
  }
  else if ((modo&0x07)==CJAVA){
   BoundingBox bounds = new BoundingBox();
   cd = new CollisionJava(shapeMama);
   cd.setSchedulingBounds(bounds);
  }
```

```
}

/**
*
* Retorna o método de deformação associado a esta classe.
*/
 public Deformation getDeformation()
 {
  return def;
 }

/**
*
* Retorna o método de colisão associado a esta classe.
*/
 public Collision getCollisionDetector()
 {
  return cd;
 }

/**
*
* Retorna o Shape3D que representa o objeto deformável.
*/
 public Shape3D getShape()
 {
  return shapeMama;
 }

/**
*
* Retorna uma cópia do Shape3D que representa o objeto deformável, utilizado no
caso de estereoscopia.
*/
 public Shape3D getStereoShape() throws Exception {
  if (shape2 == null) {
   throw new Exception ("no Estereo Shape");
  }
  else{
   return shape2;
  }
 }
}
```

Apêndice B – Código fonte da classe Object3D.java

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

/**
* Possui os atributos referentes as funcionalidades existentes, o método para adicionar os objetos no MyLocale e
* os métodos responsáveis pela transformações (escala, translação e rotação) nos objetos modelados que fazem parte do AV.
* @version 15.1 - fev 2007
* @author Ana Cláudia M. T. G. de Oliveira
*/
public abstract class Object3D {

 public static final int COLLISION = 0x01;
 public static final int OCTREE = 0x01;
 public static final int CJAVA = 0x03;

 public static final int DEFORMATION = 0x08;
 public static final int MASS_SPRING = 0x08;
 public static final int FEM = 0x018;

 public static final int STEREOSCOPY = 0x040;

 public Object3D() {
 }

 public abstract Shape3D getShape();
 public TransformGroup mTg, tg;
 private BranchGroup bg;

public boolean ehOrgao = false;

public boolean podeSerOrgao(){
   return ehOrgao;
 }

/**
* Método utilizado pela classe Environment para indicar o BranchGroup onde está localizado o Shape3D que representa este objeto.
*
*/
 public void setBranchGroup(BranchGroup bg){
  this.bg = bg;
 }

/**
* Método utilizado pela classe Environment para retorna  o BranchGroup onde está localizado o Shape3D que representa este objeto.
*
*/
```

```
 public BranchGroup getBranchGroup() {
  return bg;
 }

/**
* Método utilizado para definir uma transformação para a movimentação do objeto.
*
*/
 public void setMotionTransform(TransformGroup tg){
  mTg = tg;
 }

/**
* Método que retorna a transformação para a movimentação do objeto.
*
*/
 public TransformGroup getMotionTransform(){
  return mTg;
 }

/**
* Método que indica qual é o TransformGroup que possui os parâmetros de
posicionamento do objeto.
*
*/
 public void setTransformGroup(TransformGroup tg){
  this.tg = tg;
 }

/**
* Método define os parâmetros da escala para o objeto.
*
*/
  public void setScale(Vector3d s){
  Transform3D t3D = new Transform3D();
  tg.getTransform(t3D);
  t3D.setScale(s);
  tg.setTransform(t3D);
 }

/**
* Método define os parâmetros da translação para o objeto.
*
*/
   public void setTranslation(Vector3d t){
  Transform3D t3D = new Transform3D();
  tg.getTransform(t3D);
  t3D.setTranslation(t);
  tg.setTransform(t3D);
 }

/**
* Método define os parâmetros de rotação para o objeto.
*
*/
```

```
  public void setRotation(AxisAngle4d r){
   Transform3D t3D = new Transform3D();
   tg.getTransform(t3D);
   t3D.setRotation(r);
   tg.setTransform(t3D);
  }

/**
* Método remove um BranchGroup do MyLocale.
*
*/
 public void removeMe(){
   bg.detach();
 }
}
```

Apêndice C – Código fonte da classe Environment.java

```
package ViMeT;
import ViMeT.DefApliMedLoader.ObjectFile;
import javax.swing.*;
import java.awt.GraphicsConfiguration;
import java.awt.*;
import java.awt.event.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.geometry.*;
import com.sun.j3d.utils.behaviors.vp.*;
import com.sun.j3d.utils.behaviors.*;
import java.awt.event.*;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import com.sun.j3d.utils.behaviors.keyboard.*;
import com.sun.j3d.utils.picking.PickTool;
import javax.swing.*;
import com.sun.j3d.utils.behaviors.mouse.*;
import java.awt.TextArea;
import com.sun.j3d.utils.image.TextureLoader;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.image.*;
import java.awt.*;

import com.sun.j3d.loaders.Scene;
import com.sun.j3d.loaders.SceneBase;
import com.sun.j3d.loaders.Loader;
import com.sun.j3d.loaders.IncorrectFormatException;
import com.sun.j3d.loaders.ParsingErrorException;
import com.sun.j3d.utils.geometry.GeometryInfo;
import com.sun.j3d.utils.geometry.NormalGenerator;
import com.sun.j3d.utils.geometry.Stripifier;
import java.io.FileNotFoundException;
import java.io.StreamTokenizer;
import java.io.Reader;
import java.io.BufferedReader;
import java.io.BufferedInputStream;
import java.io.FileReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.HashMap;
import java.util.StringTokenizer;
import javax.media.j3d.*;
import javax.vecmath.Color3f;
import javax.vecmath.Point3f;
```

```
import javax.vecmath.*;
import javax.vecmath.TexCoord2f;
import java.net.MalformedURLException;
import javax.swing.*;
import java.awt.*;
import java.awt.Toolkit;
import com.sun.j3d.utils.image.TextureLoader;
import com.sun.j3d.utils.image.*;
import java.awt.image.BufferedImage;
import java.awt.image.*;

/**
 * Classe responsável pela criação do Ambiente Virtual
 *
 * @version 15.1 - fev 2007
 * @author Ana Cláudia M. T. G. de Oliveira
 */
public class Environment extends VirtualUniverse {

  /**
   * Determina a distancia entre os olhos do observador
   */
  protected float eyeOffset =0.017F;

  private Collision collision;

  Canvas3D myCanvas;

  public Locale myLocale;

  BranchGroup light;

  AmbientLight ambientLightNode;

  BranchGroup branchLamina;

  JFrame window;

 /**
   * Habilita a estereoscopia
   */
  private boolean stereoEnabled;

  /**
   * Método responsável setar a distância intraocular, utilizada na estereoscopia
   *
   *
   * @param f valor da utiizado para a paralaxe
   * default 0.017
   */
  public void setEyeOffset(float f){
    eyeOffset = f;
  }

  /**
```

```
   * Construtor responsável pela  criação dos nós Locale e todos os BranchGroup e
*TransformGroup
   * BranchdGroup:  light (responsável pela iluminação do AV); brbg (responsável pelo
background do AV);
   * leaf: DirectionalLight e AmbientLight
   *
   *
   * @param c Canvas3D utilizado
   * @param stereoEnabled Se a estereoscopia está ligada ou não.
   */
  public Environment(Canvas3D c, boolean stereoEnabled){

    this.stereoEnabled = stereoEnabled;

    myLocale = new Locale(this);
    //adicionando ao nó Locale um BranchGroup de visualização e posição
    myLocale.addBranchGraph(buildViewBranch(c));

    light = new BranchGroup();
    BoundingSphere boundsl = new BoundingSphere(new Point3d(0.0,0.0,0.0), 100.0);

    Color3f ambientColor = new Color3f(1.0f, 1.0f, 1.0f);
    ambientLightNode = new AmbientLight(ambientColor);
    ambientLightNode.setInfluencingBounds(boundsl);
    light.addChild(ambientLightNode);

    myLocale.addBranchGraph(light);

    branchLamina = new BranchGroup();

    Lamina.criaLamina(0.3f, 0.005f, 0.1f);

   Transform3D trans = new Transform3D();
    trans.setTranslation(new Vector3f(0.4f, -0.3f, 0f));
    trans.setRotation(new AxisAngle4d(2,2,1,5));

    TransformGroup transLam = new TransformGroup(trans);
    transLam.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    transLam.addChild(Lamina.lamina);

    branchLamina.addChild(transLam);

    myLocale.addBranchGraph(branchLamina);

//////////////Insere o Background na Cena

    if(!stereoEnabled){
      BranchGroup brbg = new BranchGroup();
      BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100);
      Color3f bgColor= new Color3f(1.0f, 1.0f , 1.0f);
      Background bg= new Background(bgColor);
      bg.setApplicationBounds(bounds);
      brbg.addChild(bg);
      myLocale.addBranchGraph(brbg);
    }
```

```
  }

// Método que cria o nó p/ visualização e posicionamento do ambiente
  /**
   * Método responsábel pelo subgrafo de visualização
   * BranchdGroup: viewBranch estão ligados todos os nós referentes ao subgrafo de
visualização.
   *
   *
   *
   * @param c  Canvas3D utilizado
   *
   */
  private BranchGroup buildViewBranch(Canvas3D c){
    BranchGroup viewBranch = new BranchGroup();
    Transform3D viewXfm = new Transform3D();
    viewXfm.set(new Vector3f(0.0f, 0.0f, 2.0f));

    TransformGroup viewXfmGroup = new TransformGroup(viewXfm);
    viewXfmGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
    viewXfmGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    BoundingSphere movingBounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0),
100);
    BoundingLeaf boundLeaf = new BoundingLeaf(movingBounds);

    /* Cria um objeto ViewPlataforma*/
    ViewPlatform myViewPlatform = new ViewPlatform();
    viewXfmGroup.addChild(boundLeaf);

    PhysicalBody myBody = new PhysicalBody();
    PhysicalEnvironment myEnvironment = new PhysicalEnvironment();
    viewXfmGroup.addChild(myViewPlatform);
    viewBranch.addChild(viewXfmGroup);

    /* Cria um objeto View*/
    View myView = new View();
    myView.addCanvas3D(c);
    myView.attachViewPlatform(myViewPlatform);
    myView.setPhysicalBody(myBody);
    myView.setPhysicalEnvironment(myEnvironment);

    KeyNavigatorBehavior keyNav = new KeyNavigatorBehavior(viewXfmGroup);
    keyNav.setSchedulingBounds(movingBounds);
    viewBranch.addChild(keyNav);

    MouseRotate rNav = new MouseRotate(viewXfmGroup);
    rNav.setSchedulingBounds(movingBounds);

     MouseTranslate tNav = new MouseTranslate(viewXfmGroup);
    tNav.setSchedulingBounds(movingBounds);

    return viewBranch;
  }

  /**
```

```
 * Método responsável pela adição dos objetos modelados no Universo Virtual
 * @param obj
 * @see Object3D
 */
public void add(Object3D obj){
  BranchGroup bg = obj.getBranchGroup();
  if (bg==null){
    bg = new BranchGroup();

    bg.setCapability(BranchGroup.ALLOW_DETACH);

    TransformGroup tgM = new TransformGroup();
    TransformGroup tg = new TransformGroup();

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

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

    obj.setMotionTransform(tgM);
    obj.setTransformGroup(tg);
    obj.setBranchGroup(bg);

    bg.addChild(tgM);
    tgM.addChild(tg);

    if(stereoEnabled){
      Transform3D myTrans1 = new Transform3D();
      myTrans1.setTranslation(new Vector3f(eyeOffset, -eyeOffset, 0F));
      TransformGroup tg1 = new TransformGroup(myTrans1);

      Transform3D myTrans2 = new Transform3D();
      myTrans2.setTranslation(new Vector3f(-eyeOffset, +eyeOffset, 0F));
      TransformGroup tg2 = new TransformGroup(myTrans2);

      tg.addChild(tg1);
      tg.addChild(tg2);

      Appearance ap = new Appearance();
      Color3f black = new Color3f(0.0f, 0.0f, 0.0f);
      Color3f red = new Color3f(0.7f, .0f, .15f);
      Color3f green = new Color3f(0f, .15f, .7f);
      ap.setMaterial(new Material(green,black, green, black, 1.0f));

      //textura na mama
      if(obj.podeSerOrgao() == true){

     TextureLoader loader = new TextureLoader("peleTeste.jpg", null);
     ImageComponent2D image = loader.getImage();

      Texture2D texture = new   Texture2D
(Texture.BASE_LEVEL,Texture.RGBA,image.getWidth(),image.getHeight());
      texture.setImage(0, image);
      texture.setEnable(true);
```

```
  ap.setTexture(texture);
  }
  //fim da textura na mama

 Appearance ap2 = new Appearance();
  ap2.setMaterial(new Material(red, black, red, black, 1.0f));
  float transparencyValue = 0.5f;
  TransparencyAttributes t_attr =
    new TransparencyAttributes(
                        TransparencyAttributes.BLENDED,
                        transparencyValue,
                        TransparencyAttributes.BLEND_SRC_ALPHA,
                        TransparencyAttributes.BLEND_ONE);
  ap2.setTransparencyAttributes( t_attr );
  ap2.setRenderingAttributes( new RenderingAttributes() );
  //ap.setTransparencyAttributes( t_attr );
  ap.setRenderingAttributes( new RenderingAttributes() );

  //
  PolygonAttributes attributes11 = new PolygonAttributes();
  attributes11.setPolygonMode(PolygonAttributes.POLYGON_LINE);
  attributes11.setCullFace(PolygonAttributes.CULL_BACK);
  ap2.setPolygonAttributes(attributes11);
  // ap.setPolygonAttributes(attributes11);
  //

  Shape3D shape = obj.getShape();

  shape.setAppearance(ap);
  tg1.addChild(shape);

  Shape3D shape2;
  if (obj instanceof ObjDef) {
    try{
      shape2 = ((ObjDef)obj).getStereoShape();
      shape2.setCollidable(false);
      shape2.setAppearance(ap);
      tg2.addChild(shape2);
    }
    catch(Exception e){
      System.out.println("erro: " + e.getMessage());
    }
  }
  else{
    shape2 = new Shape3D(shape.getGeometry(),ap);
    shape2.setCapability(Shape3D.ALLOW_LOCAL_TO_VWORLD_READ);
    tg2.addChild(shape2);
  }

  System.out.println("adicionado stereo - fim");
 }
 else{
  tg.addChild(obj.getShape());
 }
}
```

```
    myLocale.addBranchGraph(bg);
  }

  /**
   * Método responsável pela adição dos BranchGroup no nó MyLocale
   * @param bg
   *
   */
  public void addGrafo(BranchGroup bg)
  {
    myLocale.addBranchGraph(bg);
  }
}
```

Apêndice D – Código fonte da classe Lamina.java

```
package ViMeT;
import com.sun.j3d.utils.geometry.Box;
import javax.vecmath.*;
import com.sun.j3d.utils.image.TextureLoader;
import javax.media.j3d.*;

public abstract class Lamina extends Object3D{

  static Box lamina;

  public Lamina(){

    criaLamina(0.3f, 0.005f, 0.1f);
  }

  public abstract Shape3D getShape();

  static void criaLamina(float x, float y, float z){

    Appearance vidro = new Appearance();

    //Habilita Transparencia
    TransparencyAttributes t = new TransparencyAttributes(
                                        TransparencyAttributes.BLENDED,
                                        0.8f,
                                        TransparencyAttributes.BLEND_SRC_ALPHA,
                                        TransparencyAttributes.BLEND_ONE);
    vidro.setTransparencyAttributes(t);

    lamina = new Box(x, y, z, vidro);
  }
}
```

Apêndice E – Código fonte da classe LaminaSozinha.java

```
import java.applet.Applet;
import java.awt.BorderLayout;
import java.awt.GraphicsConfiguration;
import com.sun.j3d.utils.applet.MainFrame;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import com.sun.j3d.utils.geometry.Box;
import com.sun.j3d.utils.image.TextureLoader;

public class LaminaSozinha extends Applet {

  private SimpleUniverse u = null;

  public LaminaSozinha(){

    setLayout(new BorderLayout());

    GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();

    Canvas3D c = new Canvas3D(config);
    add("Center", c);

    BranchGroup cena = criaGrafoDeCena();

    u = new SimpleUniverse(c);

    // This will move the ViewPlatform back a bit so the
    // objects in the scene can be viewed.
    u.getViewingPlatform().setNominalViewingTransform();

    u.addBranchGraph(cena);
  }

  public BranchGroup criaGrafoDeCena() {

    BranchGroup bg = new BranchGroup();

    Appearance vidro = new Appearance();

    /* Habilita Textura
    TextureLoader loader = new TextureLoader("lamina.jpg", null);
    ImageComponent2D image = loader.getImage();
    Texture2D texture = new
Texture2D(Texture.BASE_LEVEL,Texture.RGBA,image.getWidth(),image.getHeight());
    texture.setImage(0, image);
    texture.setEnable(true);
    vidro.setTexture(texture);
    */

    // Habilita Transparencia
```

```
    TransparencyAttributes t = new TransparencyAttributes(
                                        TransparencyAttributes.BLENDED,
                                        0.8f,
                                        TransparencyAttributes.BLEND_SRC_ALPHA,
                                        TransparencyAttributes.BLEND_ONE);

    vidro.setTransparencyAttributes(t);

    Box lamina = new Box(0.3f, 0.005f, 0.1f, vidro);

    Transform3D t3D = new Transform3D();
    t3D.setRotation(new AxisAngle4d(2,2,1,5));

    TransformGroup tg = new TransformGroup(t3D);
    tg.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    tg.addChild(lamina);

    bg.addChild(tg);

    // Have Java 3D perform optimizations on this scene graph.
    bg.compile();

    return bg;
  }

  public static void main(String[] args) {
    new MainFrame(new LaminaSozinha(), 256, 256);
  }
}
```