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

LEONARDO CASTRO BOTEGA

IMPLEMENTAÇÃO DE ESTEREOSCOPIA DE BAIXO CUSTO PARA APLICAÇÕES EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

Marília
Novembro/ 2005

LEONARDO CASTRO BOTEGA

IMPLEMENTAÇÃO DE ESTEREOSCOPIA DE BAIXO CUSTO PARA APLICAÇÕES EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

Monografia apresentada ao Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como parte dos requisitos para obtenção do Título de Bacharel em Ciência da Computação.

Orientadora:
Prof[a]. Dr[a]. Fátima de Lourdes dos Santos Nunes Marques.

Marília
Novembro/ 2005

LEONARDO CASTRO BOTEGA

IMPLEMENTAÇÃO DE ESTEROSCOPIA DE BAIXO CUSTO PARA APLICAÇÒES EM FERRAMENTAS DE REALIDADE VIRTUAL PARA TREINAMENTO MÉDICO

Banca examinadora da monografia apresentada ao Programa de Graduação da UNIVEM/ F.E.E.S.R, para obtenção de conclusão do curso em Ciência da Computação. Área de concentração: Realidade Virtual.

Resultado: _____

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

EXAMINADOR 1:_____

EXAMINADOR 2:_____

Marília, 28 de Novembro de 2005

“A palavra impossível só existe no dicionário dos perdedores”

Napoleão

AGRADECIMENTOS

Primeiramente a Deus pela saúde e o privilégio de estar estudando.

Agradeço meus pais pela confiança e investimento dedicados durante todo o curso.

À minha orientadora Fátima pela excelente orientação e por agüentar minhas constantes reclamações.

À minha irmã Mariana, minha namorada Claudia e meus amigos Montanha, Larissa, Bárbara e todo pessoal dos laboratórios de pesquisa do UNIVEM, em especial à galera do LApIS.

À FAPESP (processo nº 2005/00111-0) – Fundação de Amparo à Pesquisa do Estado de São Paulo pela confiança e investimento dedicados à este projeto.

BOTEGA, Leonardo Castro. Implementação de Estereoscopia de Baixo Custo para Aplicações em Ferramentas de Realidade Virtual para Treinamento Médico. 2005. 105 F. Graduação em Ciência da Computação – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

RESUMO

A estereoscopia é um fenômeno que faz com que uma pessoa tenha sensação de profundidade ao observar uma cena em qualquer ambiente. Usando o mecanismo de visão binocular, o cérebro humano trata imagens bidimensionais para conseguir tal efeito, sendo que cada olho vê uma imagem, definindo assim a "Visão Estéreo". Neste trabalho foi desenvolvida a técnica de Estereoscopia denominada Anaglifo, utilizando-se da linguagem Java e a API Java 3D, com o principal objetivo de prover uma ferramenta rápida e prática para estudos e diagnósticos da área médica. Desta maneira, modelos importados foram trazidos ao ambiente virtual, tiveram seus posicionamentos cuidadosamente calibrados e suas cores devidamente tratadas. Conseqüentemente, a técnica implementada gera a impressão de que o modelo está em relevo em relação ao seu plano inicial, possibilitando a imersão no ambiente sintético.

Palavras – chave: Treinamento Médico, Realidade Virtual, Imagens Médicas, Anaglifos, Estereoscopia.

BOTEGA, Leonardo Castro. Implementação de Estereoscopia de Baixo Custo para Aplicações em Ferramentas de Realidade Virtual para Treinamento Médico. 2005. 105 F. Graduação em Ciência da Computação – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2005.

ABSTRACT

The Stereoscopy is a phenomenom that makes a pearson feel depth when observing a scene. Using the mechanism of binocular vision, the human brain treats bidimensional images to obtain such effect, considering that each eye sees an image, defining the "Stereo Vision". In this work, a Stereoscopy technique called Anaglyph was developed by using Java language and Java 3D API, with the main objective of providing a quick and pratical tool of Virtual Reality for medical studies and diagnostics. Thus, imported models were brought to the virtual ambient, had their positions carefully calibrated and your colors duly treated. Consequently, the implemented technique gives the sensation that the model is out of its initial plan, provoking the imersion in the sintetic ambient.

Keywords: Medical Training, Virtual Reality, Medical Images, Anagliphs, Stereoscopy.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API – *Application Programming Interfaces*

RV – Realidade Virtual

AV – Ambiente Virtual

3D – Tridimensional

J3D – Java 3D

VRML – *Virtual Reality Modeling Language*

RGB – *Red, Green and Blue*

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Um Ambiente Virtual (AV) configura-se como um ambiente tridimensional gerado por computador. Em sua grande maioria, ambientes sintéticos tratam a reprodução de situações as mais próximas possíveis da realidade, com o principal objetivo de permitir simulação, treinamento, visualização ou qualquer outro tipo de atividade através de técnicas de Realidade Virtual (RV) (MORIE,1994).

Atualmente, a Realidade Virtual pode subsidiar inúmeras técnicas para uma modelagem satisfatória e um correto tratamento de ambientes sintéticos, puros ou não. Tais técnicas possibilitam a criação de ferramentas que envolvam o usuário final de um forma mais imersiva e interativa, tratando de forma mais prática informações de processamento complexo.

Ferramentas de RV estão sendo desenvolvidas em vários centros de pesquisa do mundo, com o objetivo de auxiliar profissionais de diversas áreas, incluindo a área de saúde, no treinamento de procedimentos usuais. Como exemplo podem ser citados projetos para simulação de laparoscopia (HALUCK *et al*, 2001), ortopedia (SOURIN *et al*, 2000), endocospia (YAGEL *et al*, 1996), oftalmologia (HUNTER *et al*, 1995) e oncologia pediátrica (MACHADO, 2003).

Tais projetos para o treinamento médico necessitam de um alto grau de realismo. Para que isto ocorra é necessário que o usuário sinta-se dentro do ambiente sintético para, assim, desempenhar seus procedimentos com mais precisão. À sensação de estar dentro do ambiente sintético denominamos *imersão.* O uso de dispositivos não convencionais, como óculos estereoscópicos pode trazer um grau maior de imersão ao usuário, tentando fornecer realismo em um treinamento médico.

O cognato estereoscopia abrange todos os métodos conhecidos que utilizam o mecanismo de visual binocular do ser humano, com o intuito de gerar uma sensação de profundidade a partir da utilização de duas ou mais imagens bidimensionais da mesma representação por meio de diferentes perspectivas. A interpretação de "estéreo" refere-se ao do fato de necessitarmos de dois olhos e dois ouvidos para o total entendimento da situação, conforme (PAIVA,2004).

O termo anaglifo trata-se de todas as imagens formadas pela junção das imagens obtidas pelo olho esquerdo e direito respectivamente, visualizadas por meio de dispositivos não convencionais como óculos estereoscópicos, os quais geram a sensação de imersão do usuário em um ambiente sintético submetido a variados métodos de visualização (SANTOS,2000).

Ireland et al (1998) lembra que a palavra anaglifo vem do grego e quer dizer “cinzelado em relevo”. Para que possa ser utilizado é necessário ter um par de imagens em cores diferentes (vermelho-azul ou vermelho-verde, por exemplo), correspondente a duas perspectivas de um mesmo objeto, que só serão vistas, respectivamente, pelas lentes vermelha e pela lente azul de um óculos estereoscópico. Cada olho vê duas imagens ligeiramente diferentes.

1.1 Objetivo e Justificativa

Este projeto tem como principal objetivo implementar técnicas que proporcionem aos sistemas de RV para treinamento médico, um maior nível de realismo, facilitando, assim, estudos e diagnósticos para a área.

Uma técnica que utilize estereoscopia despontaria como uma ajuda muito bem-vinda na área médica no que diz respeito à imersão do usuário no ambiente das imagens do organismo humano, facilitando sua visualização de forma substancial.

Este projeto visa a desenvolver procedimentos para gerar imagens sintéticas estereoscópicas através das técnicas de anaglifos, que, apesar de ser uma técnica conhecida de Estereoscopia, teve sua implementação utilizando a linguagem Java, com a vantagem de um baixo custo. Para aplicar inicialmente em um Atlas Virtual 3D (MONTANHA *et al*, 2005). A principal justificativa para o desenvolvimento desta proposta é a possibilidade de proporcionar imersão e, assim, aumentar a sensação de realismo, com custo mínimo. Pretende-se, após a conclusão deste projeto, aplicar os resultados obtidos em outros projetos na área de treinamento médico.

1.2 Organização do Trabalho

Para melhor compreensão dos tópicos a serem discutidos, esta monografia está organizada em quatro capítulos, além desta introdução.

No capítulo 2 são discutidos os conceitos gerais de estereocopia, as técnicas de cada um dos métodos para obtê-la bem como o hardware necessário para sua utilização. Neste capítulo também são descritos os conceitos de anaglifo e as aplicações de estereoscopia em treinamento médico.

No capítulo 3 são apresentadas as características da API Java 3D, tratando dos conceitos de Grafo de Cena, Geometrias e Manipulação de Imagens. É apresentada também a implementação de Anaglifos em Java 3D, especificando as classes e métodos utilizados para tal fim, e, discutindo a utilidade da ferramenta para o treinamento médico.

O capítulo 4 mostra os resultados obtidos com a implementação do Anaglifo e a sua utilidade junto ao Atlas Virtual (MONTANHA, et al 2005). Neste capítulo também são vistas avaliações de usuários do sistema, mediante questionário.

No capítulo 5 são descritas as conclusões do projeto a partir dos resultados e os trabalhos futuros que podem dar continuidade ao presente trabalho.

E por fim, são apresentadas as referências bibliográficas que forneceram a base teórica do projeto.

CAPÍTULO 2 – ESTEREOSCOPIA

2.1 Conceitos Gerais

De acordo com Fontoura (2001), no curso da evolução, alguns animais (inclusive o ser humano) perderam a capacidade de enxergar em 360 graus, passando a apresentar os olhos posicionados na frente da cabeça. Desta maneira, os mesmos adquiriram um novo campo de visão, a visão binocular ou estéreo, proporcionados por olhos laterais e opostos.

Para compreender, na prática, o conceito fundamental de visão estéreo e a sua relevância no processo de reconhecimento de objetos no ambiente, basta que se feche um dos olhos e se tente fazer as atividades cotidianas. Do ponto de vista monocular, tarefas simples como se alcançar um objeto sobre a mesa passará a ser um desafio. O grande problema neste exemplo é perceber a profundidade exata e calcular a distância entre o observador e o objeto.

A visão estéreo é um dos principais mecanismos para o ser humano ter a noção de profundidade e tem este nome por precisar do uso de ambos os olhos. O olho esquerdo e o olho direito sempre vêem imagens diferentes (apesar de muito parecidas). De acordo com Santos (2000), o cérebro usa esta diferença para montar a imagem com profundidade. O resultado é a percepção de volume do que está sendo visto (FIGURA 1).

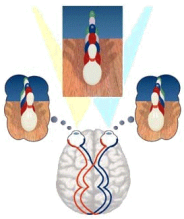


Figura 1 – Representação de Visão Estéreo (SANTOS,2002)

Paiva et al (2004) lembram que o funcionamento da percepção da profundidade foi descrito pela primeira vez por Charles Wheatstone, em 1838. A partir daí não demorou muito para que a fotografia em terceira dimensão (3D) fosse inventada. Desta forma, a mesma popularizou-se devido à adição da profundidade, a qual incrementa a sensibilidade da experiência visual, em outras palavras, o prazer de ver em três dimensões.

A interpretação tridimensional que temos do mundo é resultado da junção, pelo cérebro, das duas projeções bidimensionais captadas por cada olho a partir de seu respectivo ponto de vista e das informações sobre o grau de convergência e divergência. Segundo Fontoura (2001), os olhos humanos estão em média a 65 milímetros um do outro e podem convergir, a ponto de cruzarem suas trajetórias ficando estrábicos; podem também divergir ou ficar em paralelo quando se foca algo no infinito. Além de imagens, o cérebro coordena os movimentos dos músculos dos globos oculares e recebe as informações sobre o grau de convergência e divergência dos eixos visuais em questão, o que lhe permite calcular o ponto em que as trajetórias se cruzam em um determinado momento.

2.2 Técnicas e Dispositivos de Estereoscopia

A Estereoscopia associa-se com a capacidade de enxergar em três dimensões, isto é, de perceber a profundidade. O princípio de funcionamento da maioria dos dispositivos estereoscópicos é a projeção de imagens distintas aos olhos esquerdo e direito do observador, proporcionando sensação de profundidade, simulando uma situação real.

Algumas das principais técnicas de estereoscopia estão resumidas nas próximas seções.

2.2.1. Vídeo Estereoscópico

De acordo com Johanson (2001), a base para a percepção estereoscópica é a disparidade binocular do sistema visual humano, o qual nos fornece duas imagens ligeiramente diferentes quando uma imagem é projetada nas retinas dos olhos. Tais projeções são fundidas no córtex visual do cérebro, de forma a compor uma simples visão imersiva. Este processo pode ser simulado através de duas câmeras dispostas com a mesma distância dos globos oculares dos olhos humanos. Assim, posicionam-se as câmeras separadas uma da outra tomando como base esta distância. Quando as diferentes projeções das câmeras chegarem aos seus respectivos olhos, as duas imagens formarão uma única imagem pelo cérebro, provocando a ilusão de visão estereoscópica (FIGURA 2).

Figura 2 – Representação de “Vídeo Estereoscópico “ (JOHANSON,2001).

2.2.2. Estereoscópio

O estereoscópio é um instrumento composto por lentes que direcionam uma das imagens do par estereoscópico para o olho direito e outra para o olho esquerdo permitindo visualizar a imagem de forma tridimensional. É constituído por um par de lentes convexas montadas sobre um suporte. Ele basicamente separa fisicamente as visões esquerda e direita, eliminando a possibilidade do cruzamento entre as visões (ALVES, 1999; MACHADO, 1997).

Uma das grandes vantagens desse tipo de aparelho é permitir que o observador ajuste a distância pupilar entre as lentes, bem como ajuste à distância de visualização (FIGURA 3).

Figura 3 – Representação de “Estereoscópio” (ALVES, 1999; MACHADO, 1997).

2.2.3. Polarização da Luz

A técnica de estereoscopia por polarização da luz baseia-se na utilização de filtros, os quais fazem com que as imagens projetadas do par estereoscópico sejam polarizadas em planos verticais e horizontais. Dessa maneira o observador utiliza-se de filtros polarizadores correspondentes aos planos de projeção e vê com cada olho apenas uma das imagens projetadas. Da junção das projeções vistas por cada olho obtemos a visão estereoscópica (MACHADO, 1997).

Para comprovar o funcionamento deste método de visão estereoscópica pode-se citar a utilização de dois projetores, sendo que cada um deles fornece a imagem referente a um olho. Na frente das lentes dos projetores, são posicionados filtros polarizadores da luz projetada. Os filtros são rotacionados em 90º e o observador utiliza-se de óculos com orientações iguais a dos filtros dos projetores, também com lentes polarizadas. Desta maneira, as projeções são sobrepostas em uma tela prateada para preservar a polarização e cada olho enxerga apenas a imagem de um dos projetores, gerando o efeito estereoscópico. A Figura 4 mostra o funcionamento do método de polarização da luz.

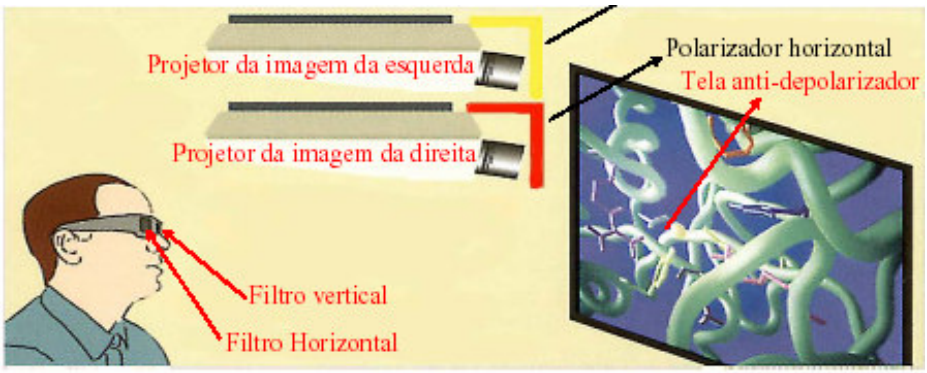


Figura 4 – Funcionamento do método de polarização da Luz(SICOUTTO *et al,* 2004)

2.2.4. Óculos Obturadores Sincronizados

Essa técnica fundamenta-se no uso de óculos especiais com lentes de cristal líquido, as quais tendem a ficar instantaneamente transparentes ou opacas de acordo com um sistema de controle prévio. Tal controle busca sincronizar sinal do vídeo com os óculos, de forma a deixar opaco ou transparente suas lentes de acordo com a imagem projetada (SISCOUTTO *et al*, 2004).

Desta maneira, a transmissão do vídeo deve apresentar, na seqüência, as imagens esquerda e direita sincronizadas. Devido à rápida atualização dos quadros, cada olho deve

enxergar uma imagem diferente, resultando no efeito estereoscópico. A Figura 5 mostra os óculos obturadores.

Figura 5 – Óculos obturador sincronizado(SICOUTTO *et al,* 2004)

2.2.5. Par Estéreo

Na visualização do par estéreo, são apresentadas duas imagens, lado a lado, de forma com que cada imagem seja posicionada levando em conta a distância entre os olhos do observador. Para o correto funcionamento deste método, o usuário deve convergir os olhos até ver três imagens.A imagem central aparece com profundidade (SISCOUTTO *et al*, 2004) (FIGURA 6).

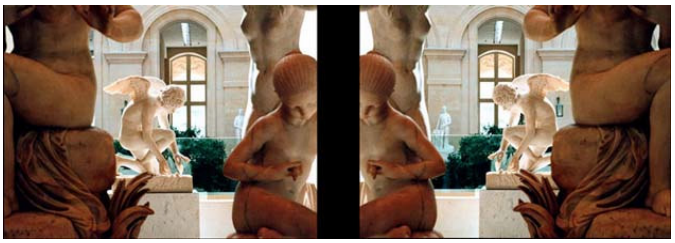


Figura 6 – Representação de Par Estéreo (LIPTON, 1982)

2.2.6. Efeito Pulfrich

Quanto menor a intensidade da luz, mais lenta é a percepção do olho humano quanto às reações aos diversos estímulos. Assim, essa técnica de estereoscopia utiliza-se de um filtro

em um dos olhos para a produção da imagem estéreo ao visualizar uma animação (FIGURA 7). A percepção diferenciada da mesma projeção pelos dois olhos estimula o usuário a enxergar o mesmo objeto em posições diferentes com cada olho, gerando o efeito imersivo. Trata-se de uma técnica extremamente simples, entretanto só funciona com modelos em movimento (SISCOUTTO *et al*, 2004).

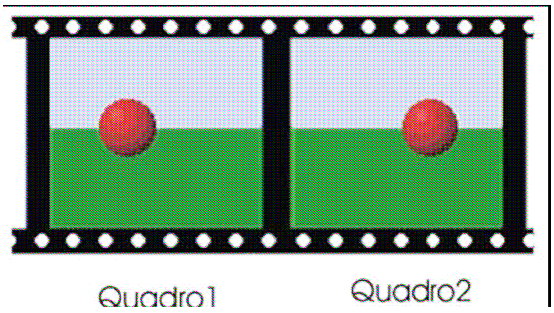


Figura 7– Efeito Pulfrich (SICOUTTO *et al,* 2004)

2.2.7. Estereograma de Pontos Aleatórios

Os estereogramas funcionam seguindo o mesmo princípio do par estéreo. Entretanto, as duas imagens são construídas sobre uma mesma projeção com apenas uma parte alterada (a qual terá a profundidade mudada) (SISCOUTTO, 2004) (FIGURA 8).

Figura 8 – Representação de Estereograma (SISCOUTTO,2004)

2.2.8. Disparidade Cromática (CromaDepth^{tm})

Ao método de estereoscopia que utiliza-se de cores e diferentes profundidades codificadas, dá-se o nome de Disparidade Cromática. Esta técnica baseia-se na utilização de lentes especias denominadas *ChromaDepth^{tm}*, as quais realizam a codificação das profundidades através de suas cores. As lentes CromaDepth mudam a trajetória da luz que as atravessa de acordo com a cor do objeto, gerando o efeito estéreo. Os objetos que possuem cores quentes (vermelhas) simulam uma maior proximidade do observador, enquanto que os objetos de cores frias (azuis) geram a impressão de maior distância. Outras cores têm sua profundidade entre o vermelho e o azul, gradativamente (FIGURA 9).

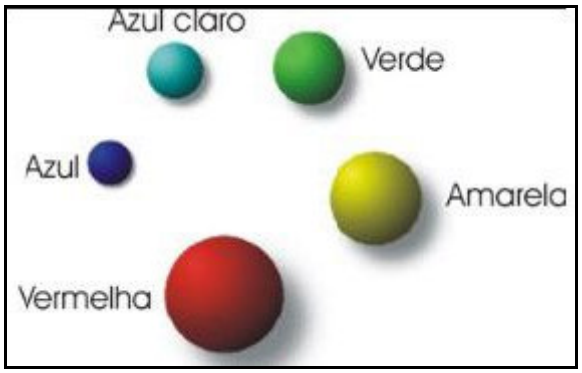


Figura 9 – Representação de Estereoscopia por Disparidade Cromática (SISCOUTTO,2004)

2.2.9. Display Autoestereoscópico

Este método de estereoscopia permite ao observador visualizar imagens tridimensionais sem qualquer tipo de óculos tridimensional. As imagens direita e esquerda são sobrepostas espacialmente em um *display*, no qual cada imagem do par estéreo é recortada e projetada sobre colunas pares e ímpares do monitor. Direcionados aos olhos do observador, os

recortes das imagens geram um efeito estéreo por meio de película colocada na superfície do monitor e pelo cálculo da distância e posicionamento da pessoa (FIGURA 10). A seguir uma tabela comparativa das técnicas de Estereoscopia (TABELA 1).

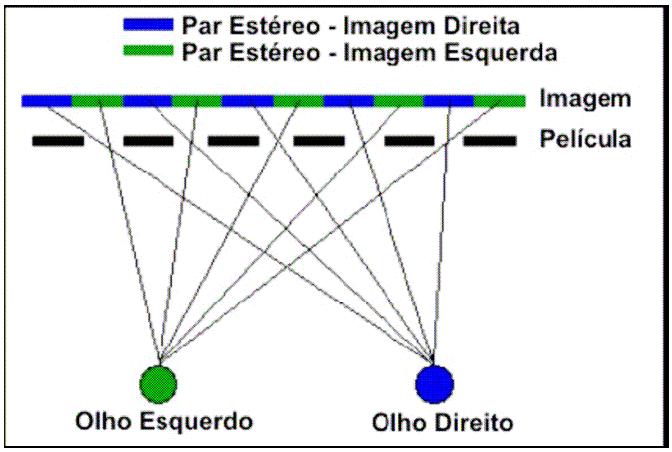


Figura 10 – Representação de Estereoscopia por Disparidade Cromática (SISCOUTTO,2004)

Tabela 1 – Comparação das técnicas de Estereoscopia

Tabela comparativa das técnicas de Estereoscopia					
	Vídeo Estereo	Estereoscópio	Polarização	Obturador	Par Estereo
Vantagens	movimento	ajuste de distância pupilar	qualidade	qualidade no monitor	baixo custo
Desvantagens	alto custo	difícil acesso	custo e dificuldade	difícil acesso	esforço ocular
	Anaglifo	Estereograma	CromaDepth	Display	Pulfrich
Vantagens	baixo custo	baixo custo	simplicidade	não utiliza óculos	simplicidade
Desvantagens	cores distorcidas	esforço ocular	qualidade em movimento	complexidade	apenas animações

Após comparar as técnicas de Estereoscopia, o método do Anaglifo foi o escolhido para desenvolver a ferramenta principalmente pelo baixo custo, visto como uma das justificativas do trabalho, que, juntamente com a linguagem gratuita Java, contribuem para tornar o produto final acessível. A seguir as características do Anaglifo.

2.3 Anaglifo

Segundo Schwartz (2004), denomina-se Anaglifo os objetos cujo produto final se obtém por uma combinação finita (azul-vermelho ou verde-vermelho) de cores, gerando,

assim, uma sensação de profundidade e imersão no ambiente (FIGURA 11). Desta maneira cada um dos olhos utiliza um filtro diferente de cor, dependendo da cor das lentes dos óculos estereoscópicos feito de papel celofane ou semelhante, para a visualização do par estéreo (FIGURA 12).

O lado vermelho do filtro refletirá apenas a cor vermelha e o lado azul/verde refletirá apenas sua cor correspondente. Conseqüentemente, as duas imagens são separadas na observação e fundidas pelo cérebro em uma única projeção tridimensional acinzentada, conforme mostra o esquema da Figura 13.

De acordo com o posicionamento dos modelos no Ambiente Virtual, o Anaglifo pode ser: (a) Invertido, quando as cores dos modelos estão posicionadas em lado contrário aos óculos do observador e; (b) Paralelo, quando as cores dos modelos estão sob a mesma perspectiva das lentes dos óculos. Neste trabalho será adotado o Anaglifo Invertido.

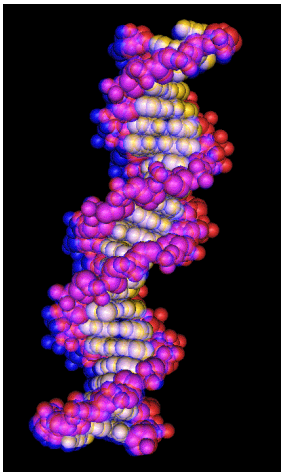


Figura 11 – Representação de um Anaglifo (SCHWARTZ,2004).

Figura 12 – Representação de óculos estereoscópico (SISCOUTTO,2004).

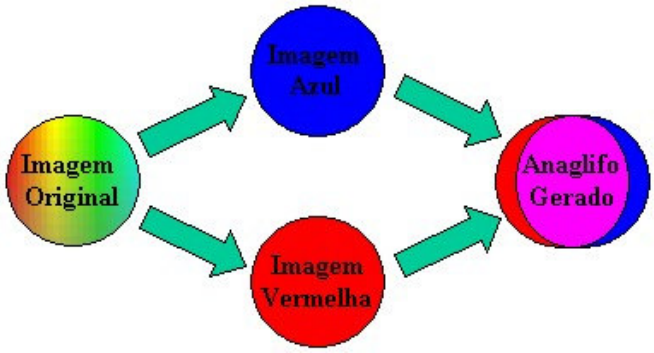


Figura 13 – Representação do funcionamento do Anaglifo (MONTANHA,2005)

As vantagens do uso de Anaglifos são a possibilidade de impressão em papel, o baixo custo, pois os óculos podem ser facilmente confeccionados e a exigência de somentes equipamentos convencionais como um monitor de vídeo ou projetor.A principal desvantagem está na perda de qualidade devido à coloração final do objeto visualizado (SCHWARTZ,2004).

Entretanto, apenas as atribuições dos pares cromáticos aos modelos do Ambiente Virtual não são inteiramente suficientes para o correto funcionamento do Anaglifo, e conseqüentemente a obtenção da imersão desejada.

Desta maneira, a transformação necessita ainda de um leve, porém necessário, deslocamento físico das visões dos modelos no eixo “X” de suas coordenadas cartesianas. Essa translação, denominada Paralaxe, torna-se necessária no sentido de suprir e simular a distância dos olhos do ser humano. Posteriormente, essas mesmas visões com seus respectivos deslocamentos laterais, serão sobrepostas por intermédio dos óculos anteriormente citado, ocultando as diferenças de posicionamento notadas sem o mesmo.

Outro aspecto necessário para a formação do Anaglifo é a questão da inclinação dos modelos de acordo com a distância focal dos olhos do observador em relação aos objetos da cena, pois além do observador não enxergar o modelo na mesma posição, as faces vistas

nunca serão as mesmas, apresentando uma rotação no eixo Y de suas coordenadas cartesianas, determinadas pela distância do observador ao objeto e seu ponto de vista.

2.3.1. Paralaxe

Conforme citado anteriormente, existem diferenças de posicionamento entre imagens formadas nas retinas de cada olho quando sobrepostas. Estas diferenças são na direção horizontal. Nas imagens estereoscópicas, a quantidade de paralaxe é a distância horizontal entre imagens esquerda e direita, determinando a distância aparente dos objetos em relação ao observador (SISCOUTTO *et al*, 2005).

Assim, disparidade e paralaxe são duas entidades similares, com a diferença que paralaxe é medida na tela do computador e disparidade, na retina. É a paralaxe que produz a disparidade, que por sua vez, produz o estéreo. Os três tipos básicos de paralaxe são: (a) Paralaxe Zero, na qual os raios de projeção de cada olho se encontram no plano de projeção, tendo a mesma projeção para os dois olhos; (b) Paralaxe Negativa, na qual os raios de projeção se encontram entre os olhos do observador e a tela de projeção, sendo que o olho esquerdo visualiza a imagem da direita e o olho direito visualiza a imagem esquerda, dando a impressão de que o modelo está em relevo na tela e (c) Paralaxe Positiva, na qual os raios se cruzam atrás do plano de projeção, dando a impressão de que o modelo está atrás da tela.

2.4. Estereoscopia em treinamento médico

As pesquisas de RV focalizando aplicações para treinamento médico crescem a cada dia, apesar de ainda serem observadas poucas publicações em nível nacional. Simuladores de procedimentos para Medicina utilizam a RV para oferecer sistemas de tempo-real interativos

e com estímulo visual, em sua maioria. A seguir são apresentadas algumas aplicações desenvolvidas com a finalidade de demonstrar as vantagens e potencialidade da estereoscopia em tais ferramentas.

O sistema para treinamento de cirurgias oculares desenvolvido na Universidade do Colorado (MAHONEY,1998) utiliza um modelo deformável do olho humano e permite determinar o local correto de incisão do bisturi com o auxílio de um dispositivo estereoscópico. Assim, o usuário pode sentir as diferentes propriedades dos tecidos que constituem o olho humano, podendo ainda visualizar um corte sendo efetuado. Os pesquisadores afirmam que a implementação de imagens estereoscópicas do olho humano, através de Java 3D, proporcionou ao usuário a sensação de estar realmente fazendo uma cirurgia, pois as imagens observadas pelo olho esquerdo e direito ao serem fundidas geram a noção de profundidade desejada facilitando assim a operação.

Meneses et al. (2002) desenvolveram um estudo para comparar as técnicas anaglífica e de polarização quanto à nitidez e realismo de um segmento neuroanatômico. Foram utilizados encéfalos completos e medulas espinhais de humanos adultos. Os mesmos foram fotografados repetidas vezes de um ponto de vista do lado esquerdo e outras pelo lado direito. Para obter as imagens em anaglifo, as imagens originais (direita e esquerda) foram sobrepostas formando um estereopar, seguido de um ajuste de deslocamento lateral, o qual foi mantido em zero para conseguir o efeito 3D e produzir um menor cansaço visual. Para correção das diferenças de luminosidade, foram trabalhadas as variações de intensidade da cor.

Para utilizar o método de polarização, as imagens esquerdas e direitas foram distribuídas em dois projetores de modo que pudessem formar um estereopar. Em seguida foram acoplados filtros polarizadores gerando a projeção simultânea das imagens e o ajuste das mesmas. O resultado foi observado com os óculos polarizadores.

Os autores concluíram que a melhor técnica em percentagem de eficácia foi a de polarização, apresentando melhor efeito 3D e maior nitidez. Esta técnica apresenta como vantagens os fatos de não haver alteração da cor natural do objeto exposto e de não necessitar de computador e programas de auxílio para seu resultado final. No entanto, há algumas limitações: não pode ser reproduzida com o uso de apenas um projetor de slides; não pode ser impressa; necessita tela especial para projeção e de filtros polarizadores para os projetores. O método anaglífico apresenta alguns pontos vantajosos: pode ser apresentado em papel impresso, utiliza apenas um projetor e não necessita de tela especial. Entretanto, há desvantagens na cor do resultado final, que fica bastante distante da original, há uma certa perda da resolução da imagem, necessita de programas de computador e scanner de dispositivos, além de gerar mais cansaço visual, segundo a opinião dos pesquisadores.

O treinamento de cirurgias ortopédicas também é tema para o desenvolvimento de simuladores. Segundo Sourin (2002), os estudantes treinam este tipo de cirurgia utilizando modelos plásticos que não possuem as mesmas densidades e características, exceto na forma, de um osso humano. A idéia de construir este simulador surgiu do fato de que modelos ósseos de boa qualidade são muito caros e do fato de que alguns tipos de ossos não possuem modelos disponíveis para compra. Com o uso de modelos de ossos virtuais com características físicas semelhantes aos reais, o sistema permite o estudo da fixação de um osso fraturado. Neste trabalho a estereoscopia foi utilizada para que se obtivesse uma melhor noção do tamanho e da forma do modelo de osso estudado. Também foi útil no processo de prática de fixação de osso fraturado com um maior grau de realismo.

Machado *et al* (2000) desenvolveram um simulador de coleta de medula óssea semi-imersivo que possibilita ao usuário treinar todas as etapas envolvidas no procedimento de coleta de medula óssea. O modelo utilizado para a simulação apresenta quatro camadas tridimensionais (epiderme, derme, osso e medula), visualizados com estereoscopia. É

utilizado um monitor associado a um óculos estereoscópico trabalhando em conjunto com um emissor infra-vermelho conectado a uma placa de vídeo e responsável pela sincronização das imagens do monitor e da obturação das lentes dos óculos. As imagens do monitor são alternadas ao mesmo tempo que uma das lentes dos óculos é obstruída. A velocidade de alternância não é visível ao usuário. Segundo os pesquisadores, neste projeto provou-se que o uso das técnicas de estereoscopia possibilitou a criação de ambientes virtuais com alto grau de fidelidade, incorporando recursos de ensino médico e avaliação de procedimentos.

O que é possível observar nesses projetos é o objetivo comum de aumentar a qualidade dos serviços médicos prestados, seja oferecendo treinamento, possibilitando um planejamento mais eficientemente de procedimentos, ou visualizando interativamente órgãos ou tecidos de difícil acesso. Projetos com o uso da estereoscopia têm demonstrado que podem promover sensações de imersão próximas dos procedimentos reais. Aplicações médicas que envolvam o profissional ou estudantes podem estimular estudos, facilitar diagnósticos e realizar cirurgias com uma maior precisão, minizando o emprego de métodos convencionais e dispendiosos, como cadáveres ou modelos sintéticos.

CAPÍTULO 3 – IMPLEMENTAÇÃO DE ANAGLIFOS

Neste capítulo são apresentadas as principais técnicas, classes e métodos disponíveis na API Java 3D para a construção de um Ambiente Virtual imersivo, bem como uma breve descrição das estruturas que compõem uma cena. Posteriormente, são apresentados os passos para a construção da ferramenta, desde a criação do ambiente até a formação do Anaglifo.

3.1 Java 3D

Segundo Manssour (2003) a API Java 3D trata-se de um pacote com classes e métodos hierárquicos ideais para a construção de interfaces destinados ao desenvolvimento de sistemas gráficos tridimensionais. Tal conjunto de classes possui construtores de alto nível que viabilizam a criação e manipulação de geometrias e objetos importados, definidos sobre um universo virtual. A mesma também possibilita a criação de ambientes sintéticos com uma grande flexibilidade e portabilidade, onde as cenas são representadas por grafos, seus atributos agrupados e suas transformações efetivadas. Desta maneira, a implementação de um programa Java 3D se resume na modelagem ou importação de modelos e na sua atribuição a grafos de cena, os quais são combinados em estruturas hierárquicas. Os grafos de cena são responsáveis pela especificação do conteúdo do universo virtual e pela forma como este é visualizado.

Java 3D foi desenvolvida pela *Sun Microsystems*, em conjunto com a *Apple*, *Intel* e *Silicon Graphics*, e teve seu código disponível a partir de 1998. Com o intuito de fornecer classes e métodos gráficos em uma plataforma independente, o Java 3D surgiu como uma alternativa de desenvolvimento semelhante ao VRML (*Virtual Reality Modeling Language*),

porém mais otimizada e baseada em tecnologias como o OpenGL e o DirectX (MANSSOUR,2003).

3.1.1. Universos Virtuais

Para a criação e manipulação de geometrias, os programadores utilizam-se de construtores de alto nível, pois os detalhes para a geração das imagens são gerenciados automaticamente. Representações 3D, juntamente com luzes, som e outros elementos integrados, compõem um universo virtual. Neste universo podem existir um ou mais grafos de cena, os quais cuidam da organização dos objetos em uma estrutura do tipo árvore (hierárquica) (MANSSOUR,2003).

3.1.2. Grafos de Cena

Instâncias de classes Java 3D que definem luz, som, orientação espacial, aparência, geometrias e localização, representam um Grafo de Cena. Tais instâncias são reconhecidas na estrutura pelos nodos (ou vértices), e os seus respectivos relacionamentos são identificados por arcos (ou arestas). As mesmas representam relacionamentos por referência, o qual simplesmente associa um objeto com o Grafo de Cena; e hierárquico (pai-filho), onde um "nodo do tipo grupo" pode ter filhos os quais contém suas definições e transformações, e apenas um pai. Um "nodo do tipo folha" não pode ter filhos. Em um Grafo de Cena habitual (FIGURA 14), os nodos do tipo grupo são identificados graficamente por círculos, e as folhas por triângulos. Assim, um círculo é considerado a raiz, e os demais são acessados hierarquicamente (MANSSOUR,2003).

Por questão de padronização, cada grafo de cena deve conter um único Universo Virtual, (objeto do tipo *VirtualUniverse)*. Tal objeto configura o universo a ser utilizado e possui pelo menos um objeto *Locale*, que demarca um ponto de referência no universo virtual e funciona como ponto base de todos os sub-grafos do tipo grupo (*BranchGroup*). A principal finalidade dos objetos do tipo grupo é associar nodos através de algum atributo comum ou por meio de um conjunto de características.

De acordo com Manssour (2003), os objetos instanciados como Grafos de Cena regem duas categorias básicas comportamentais: as que descrevem o conteúdo do universo virtual (*content branch graphs* ou sub-grafo de conteúdo), as quais tratam de aparência, geometrias, localizações, sons e iluminação, e as que controlam os parâmetros de controle da visualização da cena (*view branch graphs* ou sub-grafo de visualização).

Seguindo esta metodologia, observa-se graficamente que todos os nodos do tipo grupo são representados por círculos. Nodos folhas são identificados por triângulos e os demais objetos por retângulos.

Outros nodos de relativa importância e que serão melhores descritos no item seguinte são: *TransformGroup*, usado para especificar posições, orientações e escalas de objetos geométricos no universo virtual; *Behavior*, *Shape3D* e *ViewPlatform*. *Behavior* são "folhas" responsáveis por manipular a matriz de transformação associada com a geometria do objeto; *Shape3D* refere-se a dois objetos: *Geometry*, que trata geometrias, e *Appearance*, que possibilita a utilização de cores, texturas e transparências; e finalmente, *ViewPlatform,* instância que determina a visão final do usuário no universo sintético (MANSSOUR,2003).

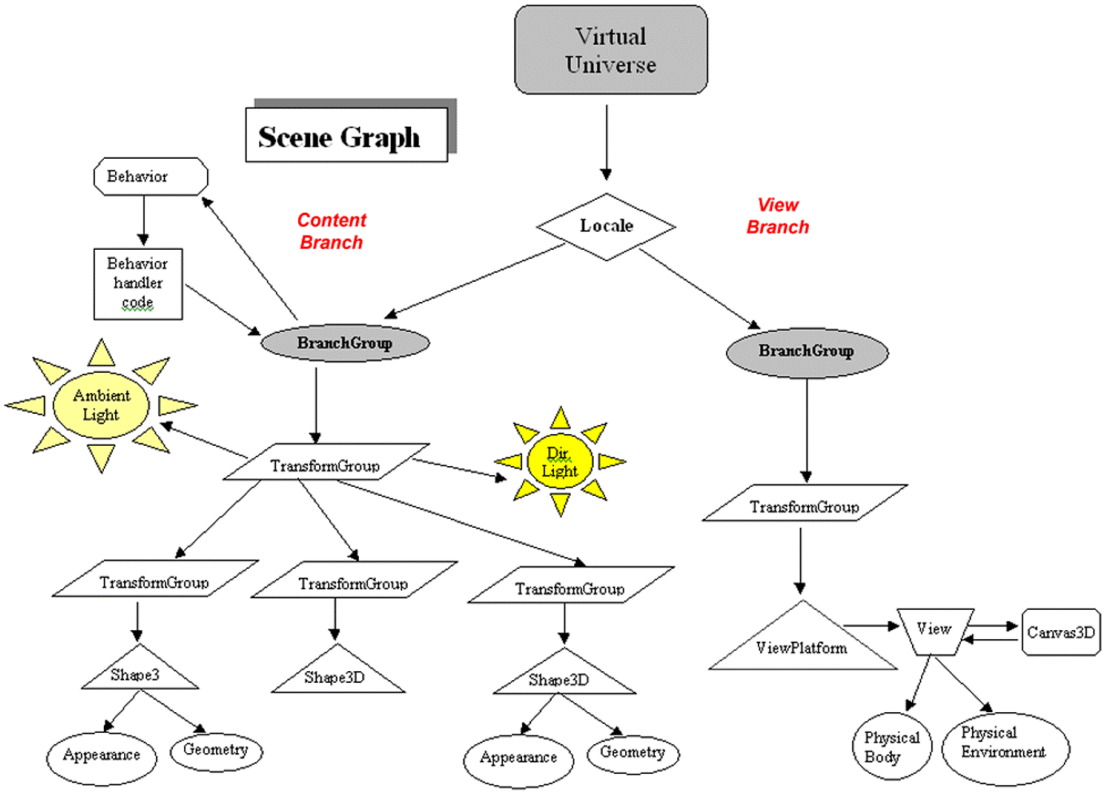


Figura 14 - Representação do Grafo de Cena (MANSSOUR,2003)

De forma simplificada, os passos para a criação de um programa Java 3D são: a criaçao de um objeto *GraphicsConfiguration* e um Canvas3D, construção e compilação de pelo menos um sub-grafo de conteúdo, criação de um objeto *SimpleUniverse*, que referencia objeto *Canvas3D* criado e criação dos objetos *VirtualUniverse* e *Locale*, construindo o subgrafo de visualização. Finalmente, insere-se o subgrafo no universo virtual.

3.1.3. Principais Classes do Java 3D

A API Java 3D possui um grande número de classes implícitas e pré-programadas para especificar, posicionar e manipular objetos gráficos modelados ou importados. Nesta seção são apresentadas algumas classes fundamentais para o desenvolvimento do trabalho aqui apresentado.

Uma classe de expressiva importância é a *SimpleUniverse*, responsável pela configuração de um ambiente propício para executar uma aplicação Java 3D, fornecendo todos os recursos necessários para a maioria das transformações. Quando uma instância de *SimpleUniverse* é criada, são criados todos os objetos necessários para o sub-grafo de visualização, tais como *Locale* e *ViewingPlatform*. Outras classes necessárias são a *GraphicsConfiguration*, a qual faz parte do pacote *awt*, responsável pela descrição das características do dispositivo gráfico (impressora ou monitor) e a classe *Canvas3D*, que fornece o *canvas*, ou seja, uma área de desenho ou trabalho, onde é realizada a visualização tridimensional (MANSSOUR,2003).

A classe *BranchGroup* serve como o ponto base de um Grafo de Cena, ou seja, a raiz das transformações. Objetos desta classe são os únicos que podem ser inseridos e associados em um objeto do tipo *Locale* , compilados, e inseridos em um universo virtual em tempo de execução.

Transformações geométricas de escala, rotação e translação, são instanciadas através da classe de *Transform3D*, formando uma matriz 4x4 de números reais (*float*). Já os objetos da classe *TransformGroup* especificam uma transformação, através de um objeto *Transform3D*, que será herdada a todos os seus filhos. Ao serem aplicadas as transformações, deve-se considerar que os efeitos num grafo de cena são cumulativos(MANSSOUR,2003). A Figura 15 mostra a hierarquia das principais classes da API Java 3D.

Já s classe *BoundingSphere* delimita as fronteiras de um objeto sob a forma de uma esfera descrita a partir de um ponto central e um raio. Tal esfera é associada com o limite do objeto, sendo usada para aplicar transformações que envolvem as dimensões da geometria.

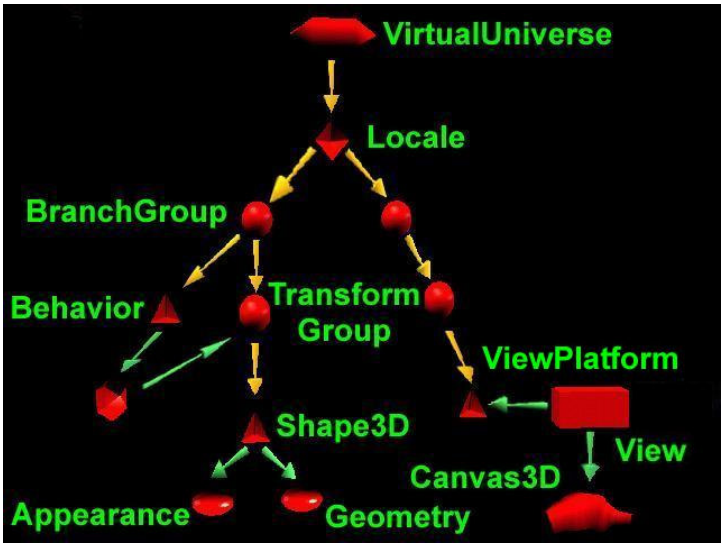


Figura 15 – Representação da hierarquia das principais classes da API Java 3D.

3.1.4. Geometrias

Por padrão, para representar entidades físicas, abstratas ou fenômenos em Computação Gráfica, são utilizados modelos. Desta maneira a modelagem passa a ser uma etapa muito importante na qual se descreve o modelo de forma que possa ser desenhado. Uma representação de um objeto deve ser feita de forma simples que facilite a usabilidade e análise.Atualmente, existem várias técnicas para a representação de modelos 3D. Nesta seção, são apresentadas formas de se representar um modelo em Java 3D, desde a sua geometria, que delimita sua forma física até a sua aparência, especificando as propriedades do material que compõe a forma geométrica, como cor, transparência e textura. (MANSSOUR,2003).

3.1.5. Representação de Objetos

Para definir estruturas básicas em Java 3D da maneira mais simples possível, utiliza-se primitivas gráficas como *Box*, *Sphere*, *Cylinder* e *Cone*, disponíveis no pacote

com.sun.j3d.utils.geometry. Sua utilização baseia-se em instâncias de classes com os mesmos nomes das primitivas.

Uma outra forma de se representar objetos graficamente é utilizando-se listas de vértices, arestas ou faces poligonais. Neste caso, malhas de polígonos representam superfícies compostas por faces planas, que podem ser triângulos (preferencialmente) ou quadrados. Desde objetos simples até os mais complexos são modelados desta maneira. O nodo *Shape3D* é usado para definir um objeto em Java 3D. Instâncias desta classe referenciam um nodo *Geometry* e um nodo *Appearance*. *Geometry* é uma superclasse abstrata que tem como subclasses, por exemplo, *GeometryArray* e *Text3D*. A classe *GeometryArray* também é uma classe abstrata, e suas subclasses são usadas para especificar pontos, linhas e polígonos preenchidos, tal como um triângulo. Algumas de suas subclasses são: *QuadArray*, *TriangleArray*, *LineArray*, *PointArray* e *GeometryStripArray*, que, por sua vez, tem *LineStripArray*, *TriangleStripArray* e *TriangleFanArray* como subclasses. Cada uma destas classes possui uma lista de vértices que podem ser conectados de diferentes maneiras. Além disso, objetos *GeometryArray* também podem armazenar coordenadas do vetor normal, de cores e de texturas (MANSSOUR,2003).

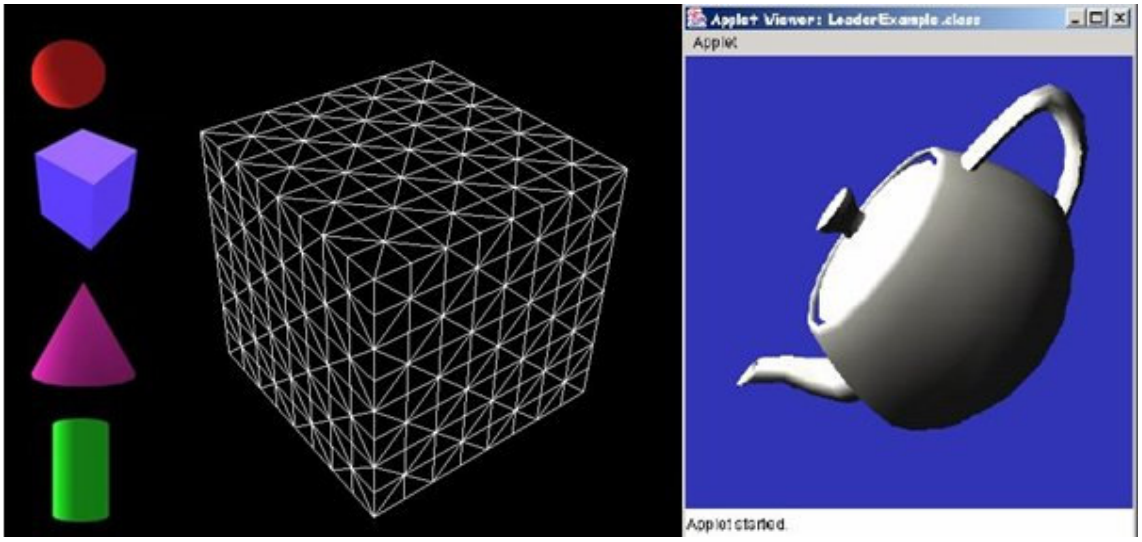
Com o Java 3D também pode-se importar dados geométricos, como conjunto de ponto, arestas ou faces, criados por outras aplicações, ou seja, trazer para dentro de um ambiente virtual, modelos desenvolvidos com ferramentas específicas. Neste caso, um arquivo de formato padrão é importado através da classe um L*oader* e a cena armazenada é representada em código Java 3D. O pacote *com.sun.j3d.loaders* fornece os subsídios para a sua implementação. Entre os arquivos que podem ser importados estão 3D Studio (.3ds), *Wavefront* (.obj), VRML (.wrl) e AutoCAD (.dfx). A Figura 16 apresenta um trecho de código que utiliza um *Loader*. A Figura 17 mostra exemplos de primitivas, listas de vértices e modelos importados.

```
ObjectFile f = new ObjectFile(ObjectFile.RESIZE,
(float)(60.0 * Math.PI / 180.0));
Scene s = null;

try
{
s = f.load( new
java.net.URL(getCodeBase().toString() + "./teapot.obj"));
}
catch (FileNotFoundException e) { … }
catch (ParsingErrorException e) { … }
catch (IncorrectFormatException e) { … }
catch (java.net.MalformedURLException ex) { … }

objRaiz.addChild(s.getSceneGroup());
```

Figura 16 – Representação de trecho de código que denota um Loader (SUN, 2005)



(a) (b) (c)

Figura 17 – (a) Representação do uso de primitivas (SUN, 2005), (b) Índices de vértices (PAVARINI *et al*, 2005) e (c) Modelos importados (MANSSOUR, 2003).

3.1.6. Aparência

Quando são criadas primitivas gráficas, as mesmas não têm cores especificadas. Tal atributo é determinado pela classe *Appearance* e, se o mesmo não for instanciado, a forma geométrica apresentará a cor branca e outras propriedades como transparência, textura e material não poderão ser manipuladas.

Alguns de seus métodos são listados na Figura 18, mostrando que esta classe faz referência a vários outros objetos.

```
void setColoringAttributes (ColoringAttributes coloringAttributes)
void setLineAttributes (LineAttributes lineAttributes)
void setTexture (Texture texture)
void setMaterial (Material material)
Material getMaterial ()
```

Figura 18 – Representação de trecho de código que demonstra a declaração de métodos da classe Appearance (SUN,2005)

Já o nodo *Material* é regularmente utilizado para definir a aparência de um objeto levando em consideração as diferentes fontes de iluminação. As vertentes a serem tratadas por esta classe são: cor ambiente refletida da superfície do material, cujo valor *default* é (0.2, 0.2, 0.2); cor difusa, que consiste na cor do material quando iluminado e possui como valor *default* (1.0, 1.0, 1.0); cor especular do material, que tem branco como *default*; cor emissiva, isto é, a cor da luz que o material emite (preto por *default*); e *shininess*, que indica a concentração do brilho do material, que varia entre 1 e 128, sendo o *default* 64. A Figura 19 mostra um trecho de código que utiliza métodos da classe *Material*.

```
Appearance app = new Appearance();
Material material = new Material(new Color3f(0.8f,0.8f,0.1f),
new Color3f(0.0f,0.0f,0.0f),
new Color3f(0.8f,0.8f,0.1f),
new Color3f(1.0f,1.0f,1.0f), 100.0f);
app.setMaterial(material);
Cone cone = new Cone(0.4f, 0.8f);
cone.setAppearance(app);
```

Figura 19 – Representação de trecho de código que demonstra o uso de métodos da classe Material (SUN,2005)

3.1.7. Iluminação

Para trabalhar com modelos iluminados, deve-se, primeiramente, definir qual a fonte de luz que será utilizada no ambiente, ou seja, o objeto que emite a energia.

Os tipos existentes de fonte de luz são: pontos de luz, luz direcional e *spot*.Uma fonte de luz por pontos é aquela cujos raios emitem energia uniformemente em todas as direções a partir de um único ponto. Uma fonte de luz direcional é aquela cujos raios vêm da mesma direção, e *spot* é uma luz que emite raios de um ponto com uma intensidade variável (SELMAN,2002).

Posteriormente e não menos importante, é preciso definir o modo com que a luz irá interagir com os modelos virtuais, em termos de superfície e natureza da luz com o principal objetivo de se obter o efeito tridimensional em espaços bidimensionais, aproximado-se ao máximo da realidade.

Os três tipos principais de modelos de reflexão são: ambiente, que é uma luz que vem de todas as direções, resultante da luz refletida no ambiente; difusa, luz que vem de uma direção, atinge a superfície e é refletida em todas as direções, fazendo com que o objeto possua o mesmo brilho independente de onde está sendo visualizado, especular, que vem de uma direção e tende a ser refletida em um única direção (MANSSOUR,2003).

Em Java 3D existe a classe abstrata *Light*, que define um conjunto de parâmetros em comum para todos os tipos de fonte de luz. Tais parâmetros são usados para definir cor da luz, se está “ligada” ou não e qual é a sua região de influência. As suas subclasses são *AmbientLight*, *DirectionalLight* e *PointLight* que, por sua vez, tem *SpotLight* como subclasse (SELMAN,2003).

A classe *AmbientLight* trata componentes de reflexão ambiente. Seus construtores permitem ativar ou não sua cor. Já a classe *PointLight* é utilizada para especificar uma fonte de luz em um ponto fixo que espalha raios de luz igualmente em todas as direções. A classe *DirectionalLight* determina uma fonte de luz regular e com destino certo com origem no infinito. Estas fontes de luz contribuem para as reflexões difusa e especular. Nos seus

construtores é possível definir a cor, a posição e o coeficiente de atenuação da fonte de luz (SELMAN,2003).

Nos construtores da classe *SpotLight* os argumentos são direção, o ângulo de expansão e concentração da luz, sendo que a intensidade da luz é alterada em função do ângulo de expansão. Os trechos de código da Figura 20 demonstram a utilização destas classes para criar diferentes efeitos de iluminação. Em cada método é utilizada uma fonte de luz diferente (direcional, pontual e *spot*) e as reflexões ambiente, difusa e especular são combinadas (MANSSOUR,2003).

```
// Exemplo de fonte de luz direcional
 Color3f corLuz = new Color3f(0.9f, 0.9f, 0.9f);
 Vector3f direcaoLuz = new Vector3f(-1.0f, -1.0f, -1.0f);
 Color3f corAmb = new Color3f(0.2f, 0.2f, 0.2f);
 AmbientLight luzAmb = new AmbientLight(corAmb);
 luzAmb.setInfluencingBounds(bounds);
 DirectionalLight luzDir= new DirectionalLight(corLuz,direcaoLuz);
 luzDir.setInfluencingBounds(bounds);
 objRaiz.addChild(luzAmb);
 objRaiz.addChild(luzDir);

 Material material = new Material(new Color3f(0.8f,0.8f,0.1f),
 new Color3f(0.0f,0.0f,0.0f), new Color3f(0.8f,0.8f,0.1f),
 new Color3f(1.0f,1.0f,1.0f), 100.0f);
```

Figura 20 – Representação de trecho de código que ilustra o uso de iluminação (SUN,2005)

3.1.8. Textura

Tipos diferentes de material possuem identificação distinta. Desta maneira, características próprias como microestruturas que produzem rugosidade na superfície dos objetos podem ser simuladas com a utilização de imagens digitalizadas específicas.

Segundo conceitos de Computação Gráfica, estas imagens da superfície de um objeto são chamadas de textura. Uma técnica de formação de texturas consiste simplesmente no mapeamento de uma imagem (mapa de textura/padrão de textura) para a superfície de um objeto (MANSSOUR,2003). Para aplicar texturas em modelos desenvolvidos através da API

Java 3D é necessário criar uma aparência, armazenar a imagem da textura e fazer a associação entre estes objetos. Também é preciso definir o posicionamento da textura na geometria, bem como os seus atributos. Resumindo, os passos para especificação de uma textura em modelagens nativas são: preparar a imagem de textura, carregar a textura, associar a textura com a aparência e determinar as coordenadas de textura da geometria.

É importante salientar que as imagens devem ser previamente tratadas com a utilização de um programa externo à API Java3D e deve estar em um formato compatível com o Java3D (JPG, GIF ou PNG). Além disso, o seu tamanho deve ser múltiplo de dois em cada dimensão. Esta imagem pode estar armazenada em um arquivo local ou em uma URL. Para carregar a textura utiliza-se uma instância da classe *TextureLoader*, que deve ser associada com um objeto *Appearance*. No final, devem-se especificar as coordenadas de textura da geometria. Nesta etapa, as posições da textura em uma geometria são determinadas por coordenadas, as quais são definidas por vértices e pontos de textura. Conseqüentemente, uma imagem pode ser esticada, rotacionada ou duplicada.

Já para modelos importados, a aplicação de texturas funciona através de um processo ligeiramente diferente, utilizando-se das classes *TextureLoader e Toolkit*, responsáveis por atribuir uma textura a um objeto do tipo *Appearance*, a qual deve ser instanciada no momento em que o modelo importado está sendo carregado, por meio do método *getImage()*, ou seja, dentro da classe *Loader*, utilizando-se de suas instâncias internas (objetos de aparência).

3.1.9. Interação

Uma interação ocorre quando o usuário recebe uma resposta, física ou meramente visual, às suas ações sobre o sistema, como pressionar uma tecla ou mover o *mouse*, cujo objetivo é mudar o grafo de cena. Tais reações são permitidas através da subclasse abstrata

Behavior que, bem como no tratamento de animações, viabiliza alterações no grafo de cena como, por exemplo, a remoção de modelos ou alguns de seus atributos (MANSSOUR,2003).

A classe *Behavior* é responsável por interpretar as ações do usuário e traduzi-las em alterações para o sistema, ou seja, faz a conexão entre o estímulo e a reação. Existe dentro desta classe, subclasses muito utilizadas para interação, por exemplo, a *MouseBehavior* e a *Keynavigatorbehavior* , que controlam eventos de *mouse* e teclado, respectivamente (MANSSOUR,2003).

Uma outra forma de interagir com os objetos em Java 3D é através dos métodos da classe *OrbitBehavior*. Desta maneira, é a *View* do ambiente que é deslocado, ou seja, os modelos permanecem estáticos enquanto é movida apenas a órbita. Fazem parte desta classe ações como rotação, translação e *zoom*. A Figura 21 apresenta um trecho de código que ilustra a utilização de métodos de interação (MANSSOUR,2003).

```
BranchGroup scene = criaGrafoDeCena();
universe = new SimpleUniverse(canvas);

ViewingPlatform viewingPlatform = universe.getViewingPlatform();
viewingPlatform.setNominalViewingTransform();

// Adiciona "mouse behaviors" à "viewingPlatform"
OrbitBehavior orbit = new OrbitBehavior(canvas,
OrbitBehavior.REVERSE_ALL);
BoundingSphere bounds = new BoundingSphere
(new Point3d(0.0,0.0,0.0), 100.0);
orbit.setSchedulingBounds(bounds);
viewingPlatform.setViewPlatformBehavior(orbit);
universe.addBranchGraph(scene);
```

Figura 21 – Representação de trecho de código que ilustra métodos de interação (SUN,2005)

3.2. Implementação do Anaglifo em Java 3D

A implementação deste trabalho foi realizada utilizando a linguagem Java com a API Java3D. A escolha desta linguagem foi devida principalmente à gratuidade de utilização,

replicação e distribuição de seus pacotes, classes e métodos, o que mais tarde será refletido no custo da ferramenta aos usuários finais, pretendendo atingir o objetivo de desenvolver uma ferramenta de baixo custo para treinamento médico. O motivo secundário da utilização desta linguagem foi devido à flexibilidade e usabilidade que a mesma fornece ao desenvolvedor.

Anteriormente a esta fase de implementação, foi desenvolvido um aprofundado estudo da API apresentada, levantando as principais classes e métodos que seriam úteis ao desenvolvimento do projeto. Paralelamente a estas atividades, foram obtidos todos os objetos modelados, ou seja, as representações do corpo humano em formato *Wavefront,* construídas através da ferramenta *3D Studio Max* (AUTODESK, 2005) e exportadas com a extensão OBJ.

Em seguida foi traçado um plano de projeto no qual foram enumeradas todas as fases a serem seguidas para a obtenção do Anaglifo, bem como a sua interface, ou seja, o modo com que a tela estaria disposta em *JFrames* (telas obtidas com a classe *Java.awt.JFrame*). Desta maneira, tais fases correspondem a módulos, cada qual representados por classes independentes, criando uma hierarquia a partir de uma classe principal com o objetivo de facilitar a manipulação e o entendimento do sistema. Tais classes independentes representam dentro do projeto, a organização interna das telas e Grafos de Cena, já que cada classe abriga o seu JFrame correspondente e suas respectivas e exclusivas transformações. As classes do sistema são apresentadas no diagrama da Figura 22. A Tabela 2 apresenta um resumo das finalidades de cada uma delas. A seguir são discutidas as funcionalidades do sistema.

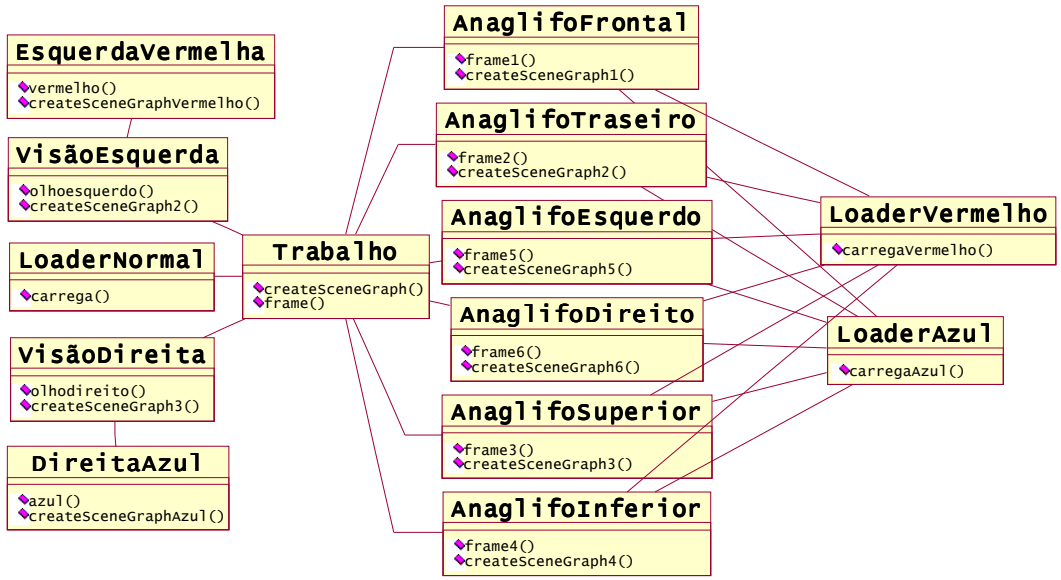


Figura 22 – Representação das classes que compõem o sistema

Tabela 2 – Representação das finalidades das classes do sistema

Classes	**Finalidade**
Loader Normal	Carrega o modelo sem transformações
Loader Azul	Carrega o modelo com a cor azul
Loader Vermelho	Carrega o modelo com a cor vermelha
Trabalho	Disponibiliza a tela inicial com o modelo sem transformações
VisãoDireita	Disponibiliza a Visão do Olho Direito
VisãoEsquerda	Disponibiliza a Visão do Olho Esquerdo
DireitaAzul	Disponibiliza a Visão do Olho Direito com a cor azul
EsquerdaVermelha	Disponibiliza a Visão do Olho Esquerdo com a cor vermelha
AnaglifoFrontal	Disponibiliza o Anaglifo carregado em sua posição inicial
AnaglifoTraseiro	Disponibiliza o Anaglifo rotacionado em 180 graus positivos no eixo Y
AnaglifoSuperior	Disponibiliza o Anaglifo rotacionado em 90 graus negativos no eixo X
AnaglifoInferior	Disponibiliza o Anaglifo rotacionado em 90 graus positivos no eixo X
AnaglifoEsquerdo	Anaglifo rotacionado em 90 graus negativos no eixo Y
AnaglifoDireito	Anaglifo rotacionado em 90 graus positivos no eixo Y

3.2.1. Visão normal do objeto

A classe Trabalho é uma classe extendida do pacote *Java.awt.JFrame* e tratada como a primeira na hierarquia do sistema, abrigando uma visão totalmente crua do objeto na cena, sem qualquer recurso estereoscópico ou de *hardware* não convencional, sendo visto pelos dois olhos simultaneamente. Tal classe abriga um *JFrame* que dispõe de um *Layout* tipo *BorderLayout*, contendo um *Canvas3D* para a exibição do objeto 3D, um *SimpleUniverse* para a viabilização do *Canvas3D,* um *BranchGroup* para abrigar o *SimpleUniverse* e uma plataforma de visualização para o grupo. São detalhados também o tamanho do *JFrame* e a posição na tela, de forma a organizar o trabalho. Em seguida é adicionado um *Container*, o qual possibilita a inserção de botões no *JFrame*, botões estes que permitirão acesso à Visão do Olho Esquerdo, Visão do Olho Direito, Obtenção do Anaglifo e Saída do Sistema; e que levarão o curso da ferramenta para outras classes, através de *Listeners*, cada qual com sua funcionalidade.

Paralelamente à criação do *JFrame*, ainda dentro da classe Trabalho, um objeto instanciado do tipo *BranchGroup* invoca o método descrito como *createSceneGraph*, o qual atuará sobre o *SimpleUniverse* criando o Grafo de Cena da etapa em questão e possibilitando a total manipulação do objeto.

O método *createSceneGraph* é iniciado com a criação de um *BranchGroup* próprio para que suas alterações sejam independentes. Em seguida é instanciado um objeto do tipo *TransformGroup* e habilitadas as funções de leitura e escrita para a manipulação de sua instância. As transformações são definidas como “filhos” do objeto *BranchGroup.*

Posteriormente, por meio de um objeto do tipo *Group*, um *Loader* é invocado, desta vez sem seu material manipulado, diferente dos próximos métodos de importação, para a

exibição da representação, pois as modelagens são desenvolvidas com a ferramenta de modelagem *3D Studio Max* e importadas para a plataforma *Java*. Também é descrito um *BoudingSphere* para a delimitação das fronteiras do objeto na cena, bem como as transformações de rotação e translação do objeto para o correto posicionamento das representações no *Canvas3D* através de instâncias do tipo *Transform3D*.

Em seguida são introduzidos, nesta compilação, os processos de iluminação ambiente e direcional, adicionados aos nodos *TransformGroup* e comportamentos de *mouse* e teclado no que diz respeito a translação e rotação executadas pelo usuário, atribuídas também aos nodos *TransformGroup* e endereçados ao "pai" *BranchGroup*. O resultado da compilação pode ser observado na Figura 23.

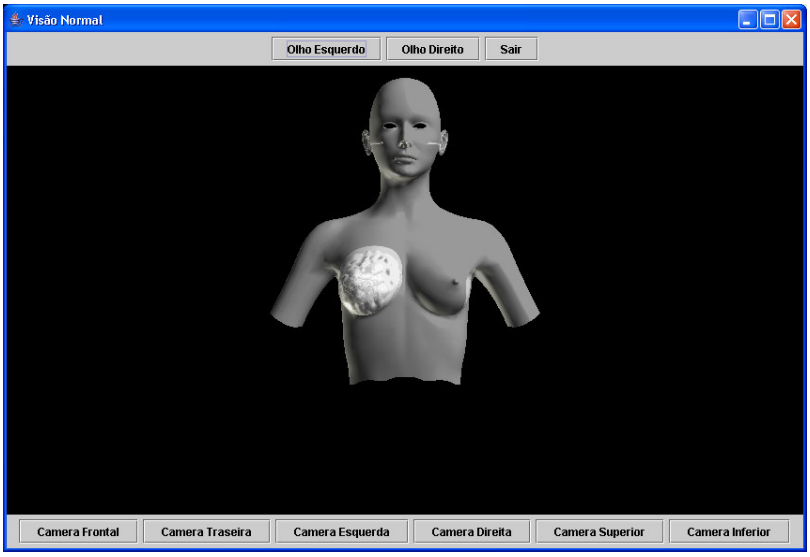


Figura 23 – Representação da Visão Normal do objeto.

3.2.2. Visão do Olho Esquerdo

A Visão do Olho Esquerdo é proporcionada pela classe Visão Esquerda, que responde ao botão "Visão do Olho Esquerdo" na tela da "Visão Normal do Objeto" e sucede-se com a chamada de uma nova classe, um nível abaixo de acordo com a hierarquia do sistema, dotada de um novo *JFrame* e outro Grafo de Cena com suas próprias características

para a simulação de como um objeto 3D seria visualizado apenas com o olho esquerdo de um ser humano.

Para a implementação da classe que representa a Visão do Olho Esquerdo foi adotada a mesma metodologia de construção da classe anterior, sendo extendida da classe *JFrame* e semelhante graficamente no que diz respeito à interface gráfica, com exceção dos botões e posicionamento da janela, diferente da anterior, para melhor observação. O único botão existente no *JFrame* desta classe é o "Camada Azul" que conduzirá o fluxo do sistema para outra classe, um nível inferior à classe atual. Aqui é tratada especificamente a Visão do Olho Esquerdo que atribuirá a cor vermelha ao objeto, filtrada pela lente vermelha do óculos estereoscópico e visualizada apenas pelo olho esquerdo na formação do Anaglifo, que será visto posteriormente.

Quanto ao *Canvas3D,* o *BranchGroup* e o *SimpleUniverse*, todos instanciados no mesmo método que constrói o *JFrame* da "Visão do Olho Esquerdo", permanecem semelhantes às suas declarações na classe que constrói a "Visão Normal do Objeto". Já o Grafo de Cena desta classe, neste ponto chamado de *create*S*ceneGraph2* sofre algumas alterações em sua estrutura, especificamente no seu posicionamento no *Canvas3D*.

O principal diferencial do *createSceneGraph2* é o tratamento do método *setRotY*, pertencente à classe *Transform3D*, tendo como parâmetro o método *math*, utilizado nesta etapa do sistema para fornecer uma inclinação, necessária à visualização do Anaglifo de aproximadamente 3 graus negativos no eixo Y, visivelmente notada observando-se o *Canvas3D* da Visão do Olho Esquerdo. Quanto às outras transformações, fronteiras e aparência, esta fase assemelha-se com Grafo de Cena da "Visão Crua do Objeto" . A Figura 24 apresenta a Visão do Olho Esquerdo.

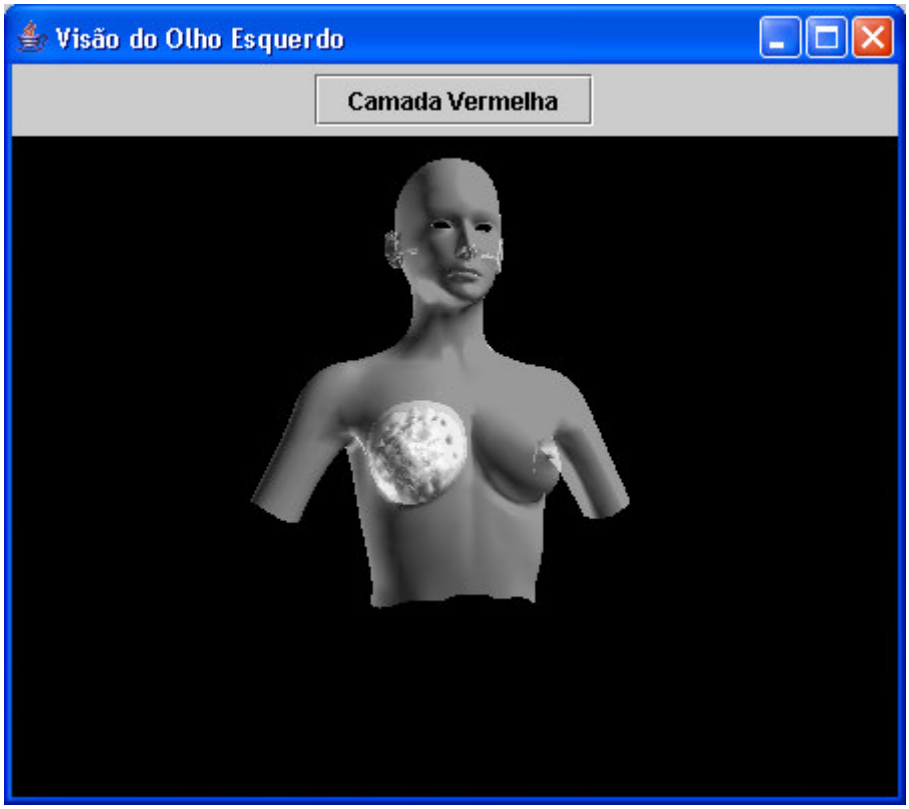


Figura 24 – Representação da "Visão do Olho Esquerdo"

3.2.3. Camada Vermelha

Seguindo a ordem de hierarquia das classes do sistema, esta etapa encontra-se um nível abaixo da classe que representa a Visão do Olho Esquerdo e tem sua execução efetivada como evento do botão "Camada Vermelha" situado na interface da classe citada.

Como nas classes anteriores, esta possui um método para a criação de um *JFrame* independente com instâncias do tipo *Canvas3D* e *SimpleUniverse*, regidos sobre uma *ViewingPlatform* e adicionados a um objeto *BranchGroup*, o qual invoca o seu próprio Grafo de Cena, contido na mesma classe. Na criação do Grafo de Cena, aqui chamado de *createSceneGraphVermelho*, foram aplicadas transformações de rotação negativa do eixo Y, idênticas às aplicadas na criação do Grafo de Cena da classe que trata da Visão do Olho Esquerdo.

O grande diferencial em nível de implementação desta classe está na atribuição da cor vermelha ao modelo, importado através da mesma classe que a Visão Normal do objeto utiliza e observada apenas pela lente vermelha dos óculos. Para conseguir tal simulação, foram utilizadas luzes ambiente *AmbientLight* e direcionais *DirectionalLight* com a cor vermelha, ao invés de aplicar a coloração ao *Material* trazido pelo *Loader*.

Em seguida são aplicados ao Grafo de Cena desta classe eventos de interação de *mouse* e teclado, como em todas as classes anteriores. A Figura 25 apresenta a Visão do Olho Esquerdo com a Camada Vermelha.

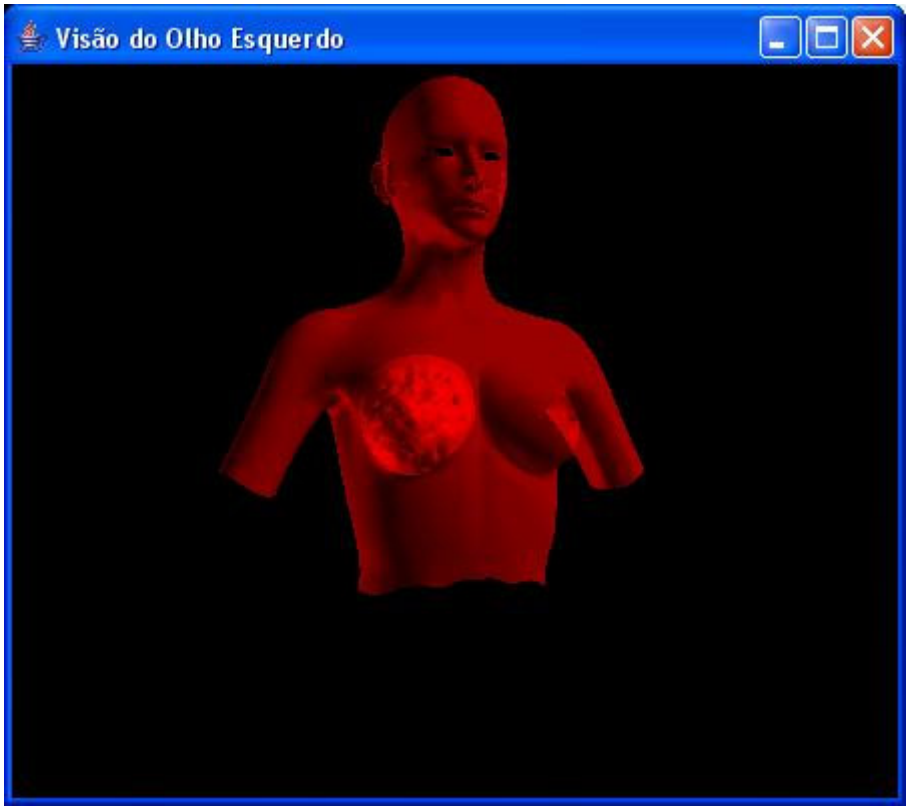


Figura 25 – Representação da “Visão do Olho Esquerdo” com a camada vermelha.

3.2.4. Visão do Olho Direito

Esta terceira etapa do desenvolvimento do sistema segue-se ao executar o botão "Visão do Olho Direito" no *JFrame* da "Visão Normal do Objeto" e sucede-se com a abertura de uma nova classe extendida da *JFrame* e situada no mesmo nível hierárquico da classe que representa a Visão do Olho Esquerdo. O desenvolvimento da mesma inicia-se através de método para a simulação de uma tela que mostra como um objeto 3D seria visualizado apenas com o olho direito de um ser humano.

Para a implementação de um *JFrame* para a "Visão do Olho Direito" foi utilizada a mesma metodologia da "Visão do Olho Esquerdo", no que diz respeito à interface gráfica, a qual também possui apenas um botão : "Camada Azul", que conduzirá o usuário a uma outra classe com o modelo que representa a visão do olho direito com sua respectiva cor, a qual será filtrada pela lente azul do óculos estereoscópico, visualizada apenas pelo olho direito na formação do Anaglifo.

Tanto para a visão do Olho Esquerdo, quanto para a visão do Olho Direito, a coloração dos modelos nesta etapa é obtida através de luz direcional ao invés de colorir o objeto em si, com finalidades meramente estéticas.

Como os *JFrames* das classes "Visão do Olho Esquerdo" e "Visão do Olho Direito" são praticamente os mesmos, com exceção do posicionamento das janelas, a diferença crucial entre as duas implementações está no Grafo de Cena, desta vez chamado de *createSceneGraph3*. O ponto que justamente distingüe as duas compilações está na transformação do eixo Y, mais especificamente o método *setRotY* do nodo *Transform3D* utilizado para fornecer o ângulo de visão do objeto visto pelo olho direito, o qual deve ser complementar ao da "Visão do Olho Esquerdo", ou seja , 3 graus positivos (FIGURA 26).

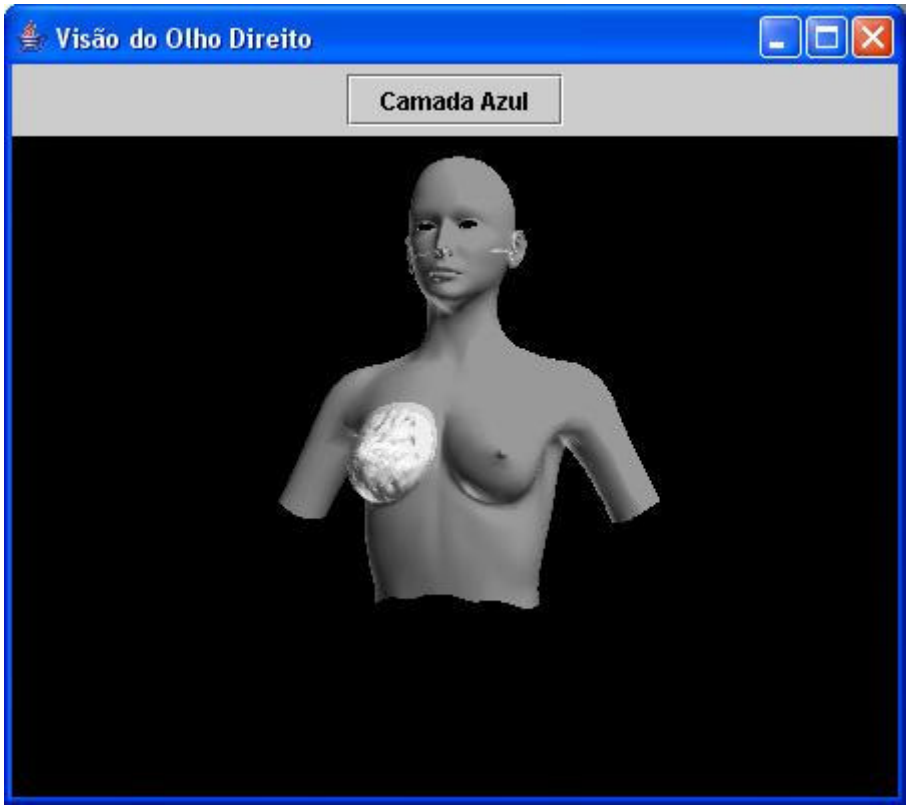


Figura 26 – Representação da “Visão do Olho Direito”

3.2.5. Camada Azul

Esta classe encontra-se no mesmo nível hierárquico da Visão do Olho Esquerdo com a Camada Vermelha e exatamente um nível abaixo da Visão do Olho Direito e tem sua execução efetivada como evento do botão “Camada Azul” situado na interface da classe citada.

Como na classe que representa a Camada Vermelha, esta possui um método para a criação de um *JFrame* independente, com instâncias do tipo *Canvas3D* e *SimpleUniverse*, regidos sobre uma *ViewingPlatform* e adicionados a um objeto *BranchGroup*, o qual invoca o seu próprio Grafo de Cena, contido na mesma classe. Ao contrário da Camada Vermelha, na criação do seu Grafo de Cena, aqui chamado de *createSceneGraphAzul*, foram aplicadas

transformações de rotação positiva do eixo Y, idênticas às aplicadas na criação do Grafo de Cena da classe que trata da Visão do Olho Direito.

Nesta etapa de desenvolvimento, ocorre a atribuição da cor azul ao modelo, importado através da mesma classe que a Visão Normal do objeto utiliza e visto apenas pela lente azul do óculos. Para conseguir tal simulação, foram utilizadas luzes ambiente *AmbientLight* e direcionais *DirectionalLight* com a cor azul, ao invés de aplicar a coloração ao *Material* trazido pelo *Loader*.

Em seguida são aplicados ao Grafo de Cena desta classe eventos de interação de *mouse* e teclado, como em todas as classes anteriores. A Figura 20 mostra a Visão do Olho Direito com a Camada Azul (FIGURA 27).

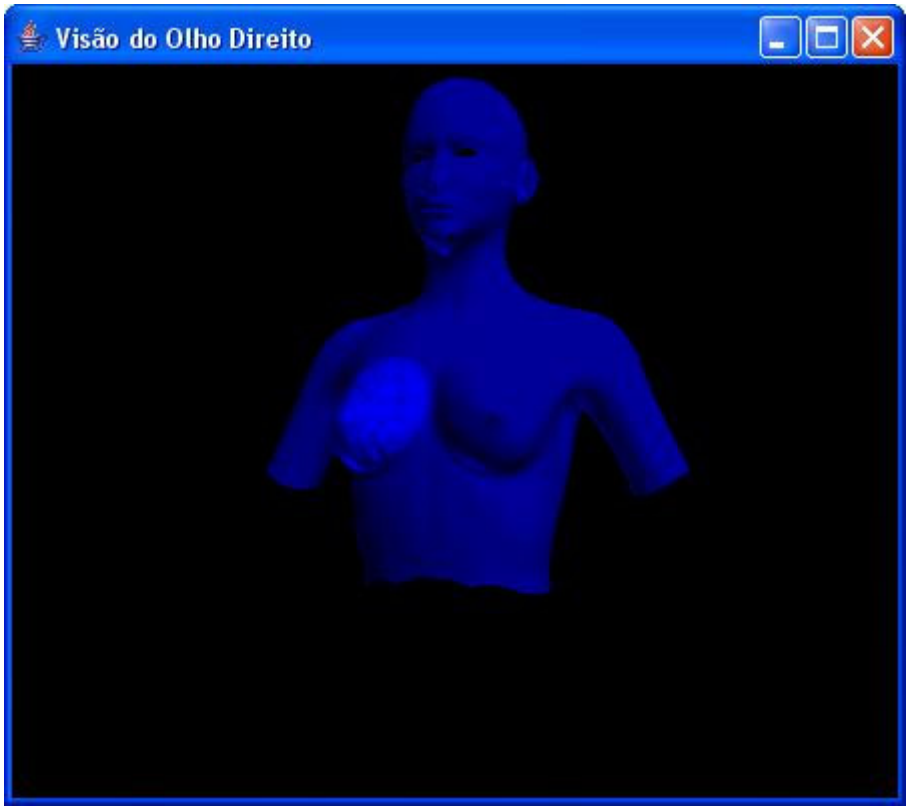


Figura 27 – Representação da “Visão do Olho Direito” com a cor azul.

3.2.6 Geração do Anaglifo

Na implementação das classes que representam o Anaglifo, ainda seguindo os mesmos padrões de telas do sistema, foi adotada a metodologia de um Anaglifo invertido, ou seja, onde os raios de projeção devem se cruzar para uma melhor visualização dos modelos. Desta maneira, a ferramenta apresenta Paralaxe Negativa, pois como será visto posteriormente, o posicionamento dos modelos quanto à rotação e translação remete à adoção desse conceito. Assim, foram fundidas as duas visões (esquerda e direita), já com as respectivas cores referentes às lentes dos óculos estereoscópicos a serem utilizados na visualização. Para viabilizar tal ação, a visão do Anaglifo foi previamente dividida em seis pontos de vista estrategicamente escolhidos fixando-se câmeras no espaço do *Canvas3D*. Conseqüentemente, foram desenvolvidos seis classes independentes, cada qual com a sua estrutura de dados, representação de tipos e *Grafo de Cena,* de modo que as configurações de um ponto de vista não interfira nos demais.

A escolha por posicionar câmeras ao redor do modelo ocorreu por razão de um detalhe particular da formação de um Anaglifo. Tanto para imagens 2D quanto para modelos tridimensionais, além da necessidade de se utilizar o par de cores azul/vermelho ou verde/vermelho para a filtragem dos óculos, necessita-se que as imagens fundidas sob o mesmo espaço virtual formem entre si um ângulo específico inclinando-se sob o eixo Y do sistema de coordenadas, variando em alguns casos particulares. Além disso, também é necessário que os modelos apresentem um deslocamento lateral, como dito anteriormente.

Desta maneira, ao posicionar um modelo no *Canvas3D* da aplicação, os modelos conjuntos devem apresentar tal angulação e deslocamento no momento em que é carregado pelo *Loader*. O problema é que apenas a visão atual apresentada e a sua complementar

apresentam tal inclinação, o que compromete a mesma ao introduzir-se eventos interativos com a rotação do modelo, perdendo a precisão do ângulo no respectivo ponto de vista.

A primeira câmera estereoscópica disponível ao usuário sucede-se como evento do botão "Anaglifo" na etapa da "Visão Normal do Objeto", podendo ser acessada a qualquer momento sem a necessidade de abertura das visões individuais. Este ponto de vista inicial trata-se da "Visão Frontal" do modelo e inicia-se com a abertura de uma nova classe, um nível abaixo da Visão Normal, e inicia-se com a criação de um *JFrame* a partir da chamada do método *frame1*( ) da classe AnaglifoFrontal, sob o qual é criado um objeto *Canvas3D*, o qual fornece subsídios para a instância de um *SimpleUniverse* que, posteriormente, irá adicionar ao Grafo de Cena um objeto *BranchGroup* através do método *addBranchGraph( )*. Posteriormente, a mesma referência do *SimpleUniverse* apóia-se em uma plataforma de visualização com o método *getViewingPlatform( )* e *setNominalViewingPlatform( )*.

Após esta etapa, ainda na criação da tela "Visão Frontal", são descritos todos os botões de acesso às outras câmeras, cada qual com a sua respectiva chamada de *frame* e conseqüente alteração nos seus Grafos de Cena, por meio de instâncias do tipo *BranchGroup*.

Todas as telas que representam os pontos de vista do Anaglifo apresentam a mesma configuração no que diz respeito à interface gráfica, ou seja, todas elas apresentam os mesmos botões que conduzem o usuário às outras classes com seus respectivos *JFrames* e Grafos de Cena.

Uma característica deve ser ressaltada na criação do método *createSceneGraphCima1* do tipo *BranchGroup,* que descreve a implementação do Grafo de Cena da classe referente à Visão Frontal: são atribuídas aos objetos do tipo *TransformGroup*, declarados anteriormente como globais, propriedades de transformação denominadas *Capabilities*, as quais permitem as ações sobre os objetos da classe de transformação.

Em seguida, as instâncias *TransformGroup* são inseridas como filhas do objeto *BranchGroup*, seguindo a hierarquia do Grafo de Cena, permitindo, assim, ações conjuntas sobre as duas. Posteriormente, são criados objetos *Group*, os quais receberão os *Loaders* e em seguida atribuídos como filhos dos objetos *TransformGroup*.

As ações de transformação começam com a declaração de dois objetos *Transform3D*, referentes às duas visões que serão incorporadas ao Anaglifo. Desta maneira, através do método *rotY( )* da mesma classe, com o parâmetro de Π/60 para o modelo vermelho e -Π/60 para o modelo azul, são ajustadas as inclinações do eixo Y dos modelos para a formação do ângulo e através do método *setTranslation( )*, com os parâmetro de 0,005 e –0,005 para as representações de vermelho e azul respectivamente, são ajustados os deslocamentos representando a distância ocular do observador. Ambos são atribuídos aos objetos *TransformGroup*.

Como dito anteriormente, os modelos apresentam inclinação de três graus positivos para o vermelho e três graus negativos para o azul, além das translações anteriormente citadas. A obtenção desses valores exatos deve-se ao fato da realização de testes práticos com o anaglifo com usuários diversos e valores variados, obedecendo aos conceitos de Anaglifo Invertido e Paralaxe Negativa.

O próximo passo é delimitar as fronteiras dos modelos através da classe *BoudingBox* para viabilizar o processo de iluminação, tando direcional quanto ambiente.

Assim, para facilitar a visualização do Anaglifo, são preparadas luzes do tipo *AmbientLight*, produzindo luz ambiente vermelha para o modelo visto pelo olho esquerdo e azul para o modelo que representa a visão do olho direito, ambas atribuídas aos repectivos objetos *TransformGroup*. Em seguida são instanciados os objetos *DirectionalLight* para incidir luz direcional branca apenas para efeitos estéticos, também atribuídos aos nodos *TransformGroup*.

Como foram posicionadas câmeras ao redor do modelo que representa o Anaglifo, os eventos interativos de rotação e translação, aplicados através do *mouse*, foram retirados dando lugar apenas à aproximação e afastamento por meio do teclado, realizados por meio de uma instância da classe *KeyNavigatorBehavior* e atribuída ao objeto *BranchGroup*, possibilitando ao usuário ver de perto as estruturas como um todo em modo estereoscópico. Ao final, o objeto *BranchGroup* é compilado pelo método *compile( )* da mesma classe e retornado para o escopo do sistema. A seguir exemplos de algumas visões do Anaglifo, como a Visão Frontal do Anaglifo (FIGURA 28), Visão Traseira (FIGURA 29), Visão Esquerda (FIGURA 30) e Visão Direita (FIGURA 31).

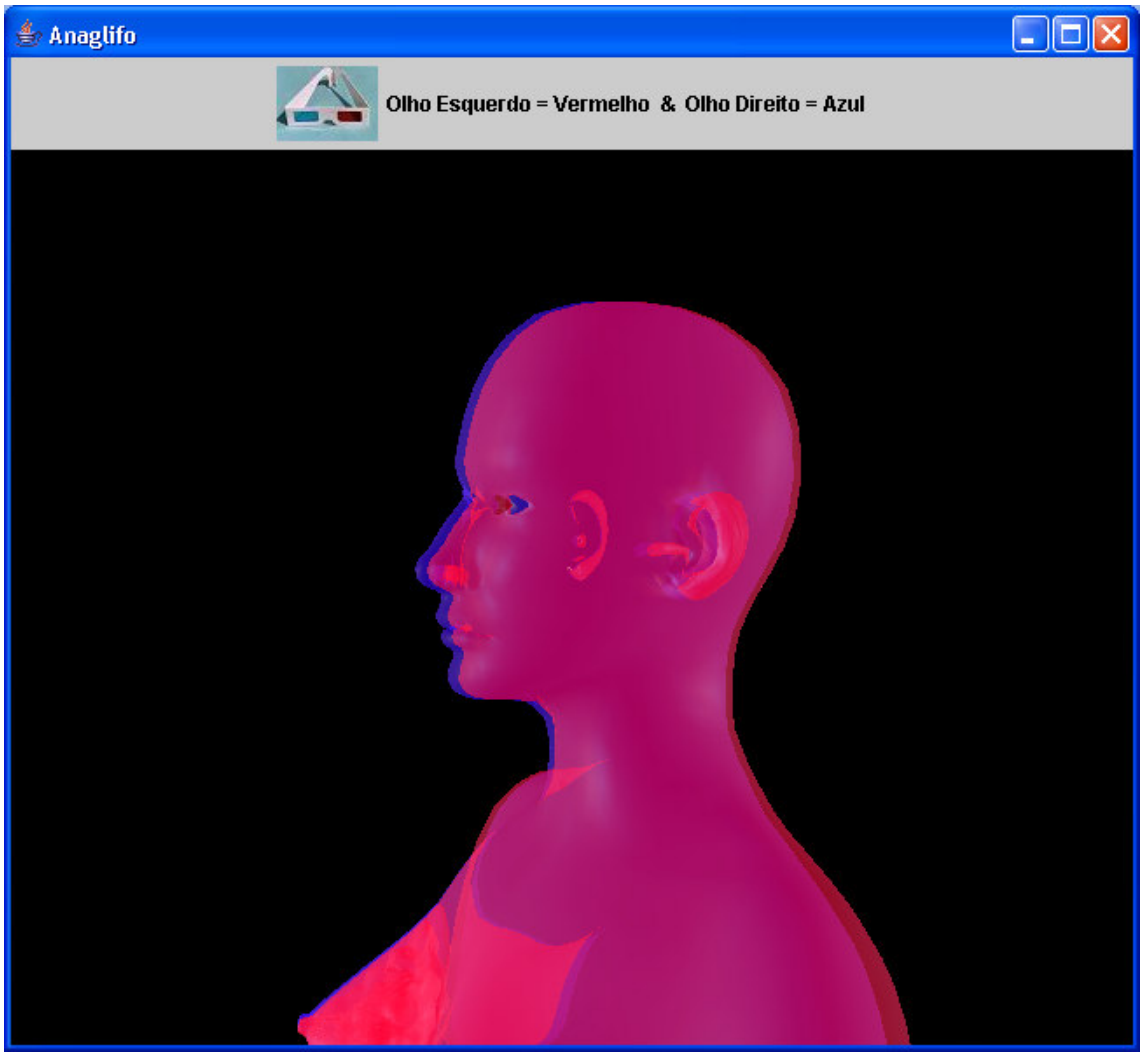


Figura 28 – Representação da Visão Frontal do Anaglifo

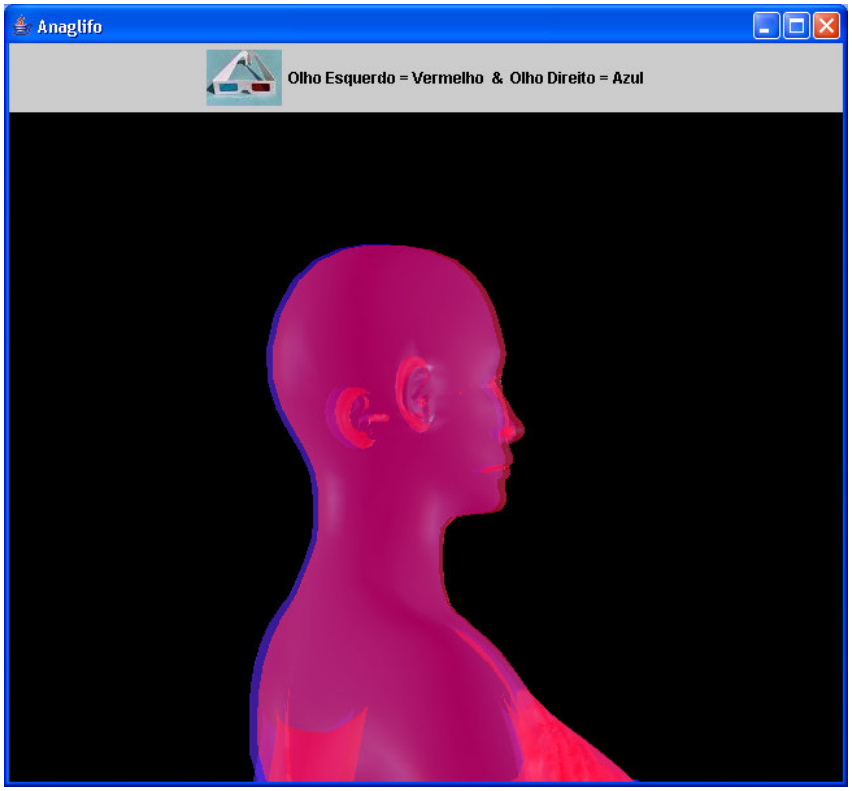


Figura 29 – Representação da Visão Traseira do Anaglifo

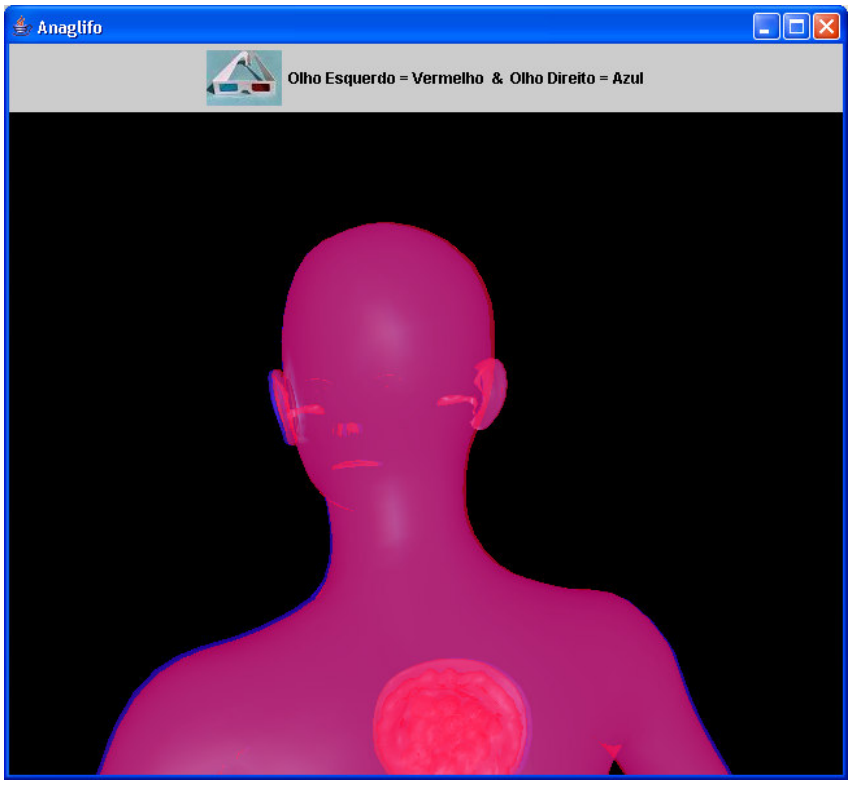


Figura 30 – Representação da Visão Esquerda do Anaglifo

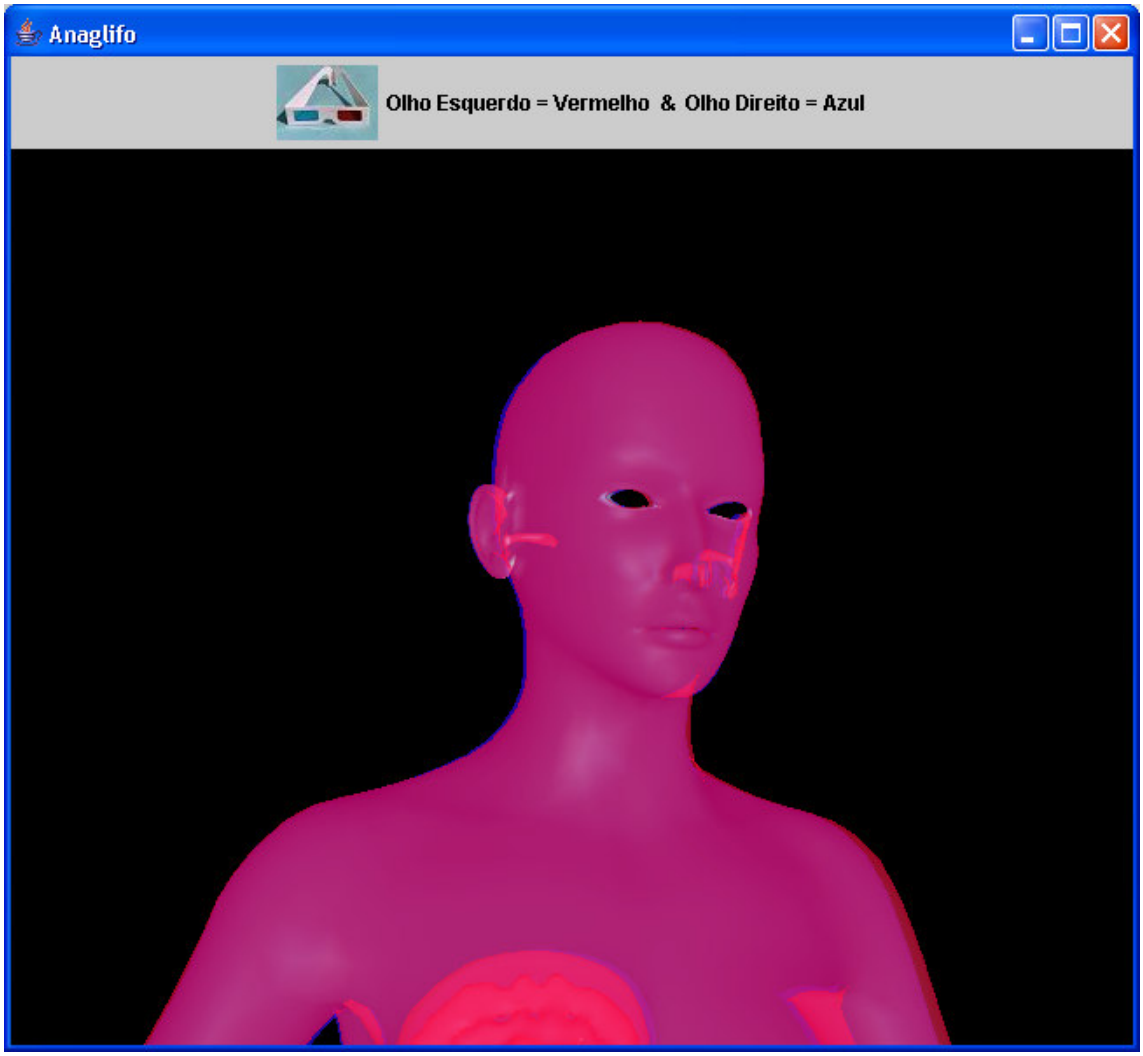


Figura 31– Representação da Visão Direita do Anaglifo

CAPÍTULO 4 – Resultados e Discussões

Neste capítulo são descritos todos os resultados obtidos com a implementação do Anaglifo em Java 3D, bem como discussões sobre métodos e parâmetros utilizados para atingí-los. Paralelamente, é apresentada uma avaliação da ferramenta, aplicada a usuários finais, bem como sua integração em um Atlas Virtual (MONTANHA, 2005).

4.1. Atlas Virtual

Para verificar o efeito da implementação descrita na seção anterior foi utilizado um projeto (MONTANHA, 2005) que visa a construção de um Atlas Virtual Tridimensional da estrutura mamária, utilizando-se microcomputadores com processador *Pentium* 4, 512 *Megabytes* de memória RAM e 128 *Megabytes* de memória de vídeo. O principal objetivo de tal trabalho é possibilitar a construção de uma ferramenta tridimensional e interativa como forma de contribuir com ensino de anatomia e fisiologia a estudantes de Medicina que, convencionalmente, é realizando através de livros e cadáveres.

Os objetos foram modelados através da ferramenta 3D Studio Max e exportados com a extensão .OBJ, sendo anexados ao Atlas Virtual já sob a forma de uma ferramenta que possibilite a visualização do corpo humano através de Anaglifo.

Entretanto, a importação de objetos pela API Java3D pode ser um problema, considerando que nem sempre as características de modelagem do aplicativo original são mantidas. Um exemplo de problema enfrentado é a dificuldade encontrada para alterar atributos, como transparência e cor, nos modelos importados. Para solucionar esta questão foi necessário manipular características do objeto no momento da importação, utilizando-se

métodos dos nodos *Appearance* e *Material* sobre um novo objeto do tipo *Shape* dentro da classe *Loader*.

Um outro aspecto que merece discussão é sustentação do posicionamento correto dos modelos obedecendo aos devidos ângulos de visão e coordenadas de translação, que são princípios básicos para a formação de um anaglifo. Essa questão foi resolvida posicionando-se câmeras em pontos geograficamente escolhidos como pertinentes ao redor do Canvas3D, deixando a cargo da aplicação a escolha dos pontos de vista a serem visualizados pelo usuário. Tal decisão, embora comprometa a interação, pois foram retirados os eventos de *mouse* como rotação e translação, fornece posições para observação do anaglifo a partir de vários pontos de vista.

Para resolver essa questão, preservando a interação na ferramenta, seria necessário desenvolver um método que aplicasse automaticamente nos modelos, os valores escolhidos para rotação e translação. Assim, toda vez que os modelos tivessem suas posições alteradas, os parâmetros do nodo *Transform3D* se adaptariam à nova posição, ajustando o ângulo e o Paralaxe.

Ainda assim, a preservação de eventos de mouse e teclado, levando em conta a complexidade dos modelos e suas transformações, não contribuiria substancialmente com a interatividade do sistema, já que a atualização do *Canvas* exigiria muito dos recursos de *Hardware* e a resposta do sistema seria lenta. Para que os modelos fossem rotacionados e transladados pelo usuário, seria necessário que fossem carregados novamente, já na posição final escolhida pelo mesmo, com seus respectivos deslocamentos e inclinações, podendo provocar uma latência na visualização.

No momento da implementação das duas visões, esquerda e direita, foi adotado um sistema de coloração baseado em iluminação, com luzes ambiente e direcionais. Contudo, este processo foi adotado devido a efeitos meramente estéticos causados sobre os modelos e não

foi possível aplicar sobre os objetos que compõem o anaglifo. Todo objeto na cena possui *bounds*, ou fronteiras, que são específicas e individuais, definindo os limites de aparência e material. Se fossem aplicadas luzes para colorir os modelos, as mesmas se misturariam e seriam geradas cores distorcidas, já que as fronteiras se misturam ao anexarem-se aos modelos. Conseqüentemente, as cores foram aplicadas sobre objetos do tipo *Material* dentro das classes que implementam *Loaders*, e posteriormente, atribuídas a um *Shape*.

As corretas colorações dos modelos foram obtidas após um longo período de testes práticos sobre o anaglifo, visto que a ferramenta não funciona com qualquer tom de vermelho e azul, devendo coincidir com as cores das lentes dos óculos. Cores fora do padrão desejado não são corretamente filtradas pelas lentes dos óculos, pois não se misturam como deveriam, impedindo a separação das mesmas e comprometendo o efeito estereoscópico. Por exemplo, tons mais escuros para o modelo vermelho formam uma sombra sobre o modelo azul, dificultando sua visualização. Desta maneira, foi adotado o sistema de cor emissiva através do método *setEmissive* da classe *Material*, que abriga parâmetros RGB, simulando a luz que se origina do próprio objeto. Esta não é afetada por qualquer outra fonte de luz e também não introduz luz adicional na cena, adaptando-se às condições que o Anaglifo requer. A Tabela 3 apresenta alguns valores testados para os parâmetros de cor dos modelos, sendo o resultado "sombra" para quando os óculos não filtrar corretamente um dos modelos e "brilhante" para quando um dos modelos ofuscar a filtragem do outro.

Tabela 3 – Valores de teste para as cores dos modelos

Valores para o modelo Vermelho							
Especular	100	100	255	200	50	20	0
Difusa	255	200	55	55	50	20	0
Emissiva	100	200	65	65	100	165	165
Valores para o modelo Azul							
Especular	100	100	255	200	50	20	0
Difusa	255	200	55	55	50	20	0
Emissiva	100	200	65	65	200	255	255
Resultado	brilhante	brilhante	brilhante	brilhante	sombra	sombra	ideal

Com os valores citados na Tabela 3, pode-se observar que cores muito brilhantes, como as produzidas pelos métodos *setSpecular* e *setDiffuse*, interferem no processo de formação do Anaglifo, ofuscando e confundindo o observador. Cores muito escuras ou opacas também contribuem para a má formação do Anaglifo, decorrente de valores baixos no parâmetro do método *setEmissive*.

Paralelamente a este problema, testes comprovaram sucintas possibilidades na angulação e deslocamento lateral dos modelos, no que diz respeito ao correto funcionamento da ferramenta. A Tabela 4 apresenta os valores testados para a rotação e translação dos modelos.

Tabela 4 – Representação dos testes de rotação e translação

Valores de Rotação e Translação testados para o modelo Vermelho						
Rotação	π/45	π/45	π/47	π/45	π/45	π/60
Translação	0,01	0,001	0,0016	0,005	0,008	0.005
Valores de Rotação e Translação testados para o modelo Azul						
Rotação	-π/45	-π/45	-π/47	-π/45	-π/40	-π/60
Translação	-0,01	-0,001	-0,0016	-0,005	-0,008	-0.005

Com os valores testados para Rotação e Translação dos métodos *RotY* e *setTranslation*, respectivamente, ambos da classe *Transform3D*, pode-se observar que quanto maior a distância entre modelos (Paralaxe), maior é o relevo aparente do modelo em relação ao plano, sendo a distância escolhida (em destaque) a que mais se adaptou aos olhos dos usuários.

O mesmo acontece para a inclinação dos modelos a respeito do conforto aos olhos do usuário. Os valores destacados na Tabela 4 referem-se aos que melhor se ajustaram na visualização. Visto que o ângulo formado na junção das duas visões baseia-se na posição do

observador e depende única e exclusivamente desta constante, os valores foram escolhidos levando em conta a distância de 30 centímetros do observador à tela do monitor de vídeo e de acordo com a adaptação às translações, considerando que o modelo foi carregado na origem das coordenadas cartesianas para os três vetores.

Com a ferramenta finalizada, foi possível obter resultados satisfatórios no que diz respeito ao realismo e à imersão definidos como metas do projeto. O anaglifo implementado gera a impressão de que o modelo está “em relevo na tela” cumprindo com os objetivos propostos.

A respeito de melhorias, a ferramenta poderia dispor de um sistema adaptável às diferentes variações de lentes dos óculos estereoscópicos, já que nem todos os dispositivos possuem a exata coloração dos modelos. Não menos importante, a ferramenta também poderia dispor de eventos de interação, como rotação e translação, integrados às cenas dos Anaglifos, fornecendo ao usuário uma resposta mais adequada do sistema. Posteriormente, poderiam ser somados ao sistema, várias outras classes que implementem outros métodos de Estereoscopia, enriquecendo o projeto.

4.2. Avaliação de Usuários

Para melhor avaliar a performance, utilidade e facilidade do sistema construído foi realizada uma pesquisa junto a um grupo de doze usuários que têm alguma ligação com o desenvolvimento de ferramentas para treinamento médico. Foi solicitado a eles que se realizassem as tarefas permitidas. Após a utilização foi preenchido um questionário (APÊNDICE O).

Os principais resultados são apresentados a seguir. A primeira parte do questionário tinha o objetivo de identificar os usuários em relação à idade, nível de escolaridade, profissão e área de atuação.

A segunda parte do questionário distribuído consistiu na avaliação do sistema propriamente dito. Foram elaboradas seis questões objetivas e uma questão aberta, na qual o usuário poderia tecer considerações gerais sobre o sistema. A seguir os gráficos que representam as respostas dos usuários ao questionário.

Na Figura 32, que representa a porcentagem da capacidade de entendimento do sistema, pode-se observar que 80% dos usuários que testaram o sistema, compreenderam-no plenamente. A respeito da facilidade de acesso às funções do sistema, 70% dos usuários conseguiram efetuar a tarefa com sucesso, sendo que apenas uma pessoa achou ruim a *interface* (FIGURA 33). Quando questionados sobre a visualização de apenas um modelo estéreo, a maioria conseguiu suficiente identificação (FIGURA 34).

Como pode ser observado na Figura 35, 55% dos usuários acharam satisfatória a ferramenta quanto à percepção do modelo em relevo na tela. Apenas 5% não verificaram efeito algum e 35% acharam o efeito regular. A respeito da identificação de apenas uma borda, a maioria dos usuários questionados acha que tal processo necessita de reparos (FIGURA 36). Entretanto, levando em conta os resultados finais, 75% dos usuários consideram que a ferramenta está apta para auxiliar o treinamento médico (FIGURA 37).

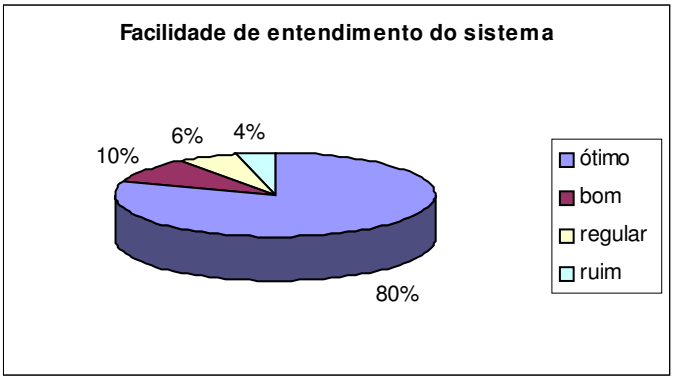


Figura 32 – Gráfico que representa a facilidade de entendimento do sistema

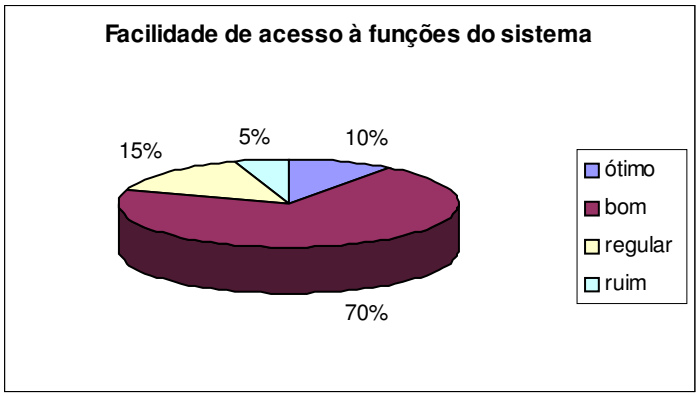


Figura 33 – Gráfico que representa o grau de facilidade de acesso às funções

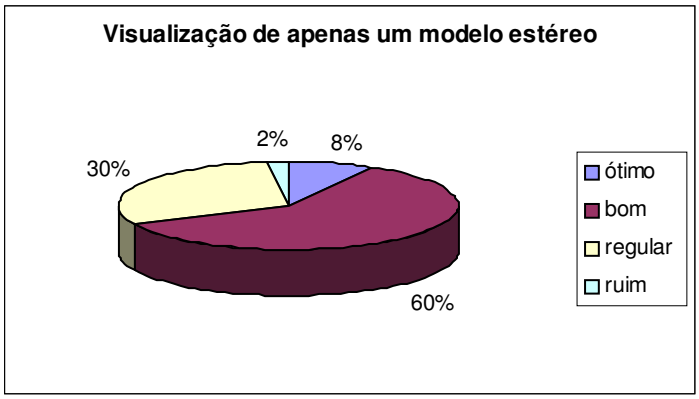


Figura 34 – Gráfico que representa a visualização de apenas um modelo estéreo

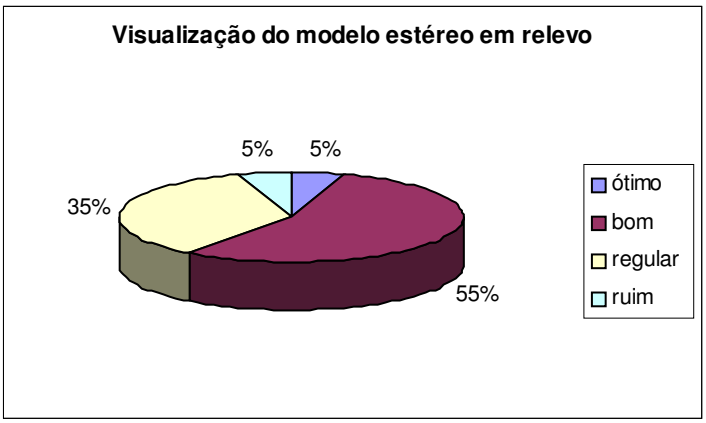


Figura 35 – Gráfico que representa a visualização do modelo estéreo em relevo pelos usuários

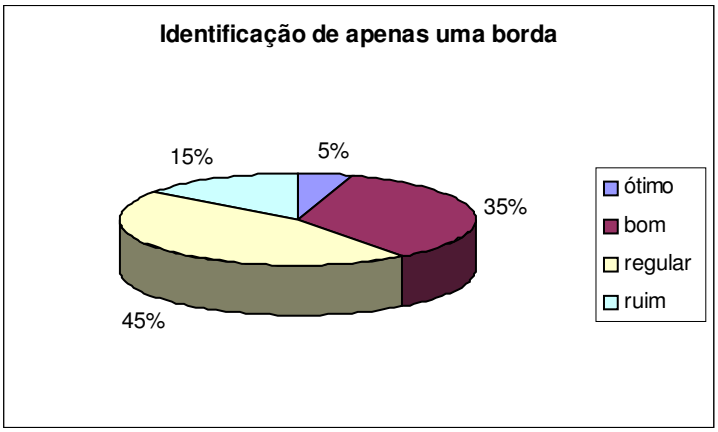


Figura 36 – Gráfico que representa a identificação de apenas uma borda pelos usuários

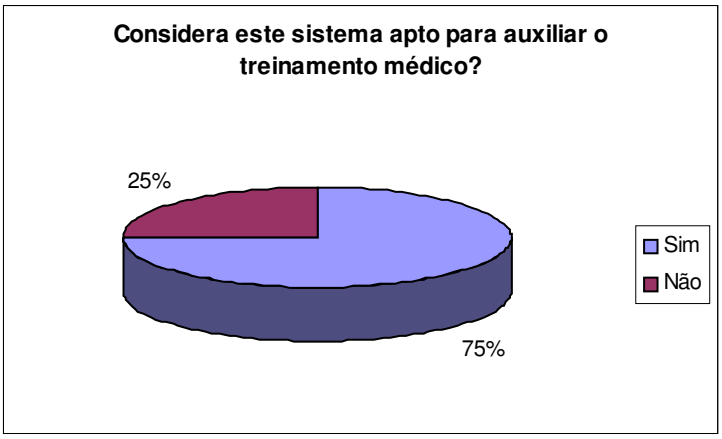


Figura 37 – Gráfico que representa o grau de aceitação do sistema.

Posteriormente aos testes realizados com desenvolvedores de aplicações para treinamento médico, a ferramenta deverá ser testada com usuários da área médica. Desta maneira, a eficácia da mesma poderá ser mais eficientemente medida e possíveis falhas poderão ser devidamente indicadas para correção.

CAPÍTULO 5 - CONCLUSÕES E TRABALHOS FUTUROS

Apesar do fornecimento de métodos para implementação de estereoscopia pela linguagem Java e pela API Java 3D, verifica-se que estes são construídos dentro de conceitos pré-estabelecidos, sem flexibilidade. Aproveitando-se tais métodos, novas classes foram construídas a fim de permitir que uma aplicação possa implementar estereoscopia com anaglifos de forma fácil e rápida, gerando o efeito de imersão desejado, sem precisar preocupar-se com as características de geração, armazenamento e recuperação das imagens estereoscópicas. Assim, os resultados iniciais deste trabalho mostraram que é possível utilizar tecnologias gratuitas para construção de ambientes virtuais com custo mínimo.

Os resultados iniciais obtidos com o sistema foram satisfatórios após uma série de testes práticos feitos com estudantes de computação, confirmando a pertinência da aplicação no que diz respeito à imersão em ambiente virtual. Entretanto, o sistema deverá passar por testes mais exaustivos executados por usuários da área médica para a detecção de possíveis falhas e confirmação da usabilidade.

A implementação de outras técnicas de estereoscopia surgiria como um expressivo ganho na qualidade do sistema, tornando-o ainda mais abrangente e portável para a utilização em outras ferramentas de treinamento da área médica. Um ponto que merece destaque é que a utilização de dispositivos não convencionais, como óculos obturadores, pode encarecer projetos, podendo inviabilizar o uso de ferramentas virtuais de treinamento médico em locais com pouco ou nenhum recurso. Assim, através deste trabalho foi possível perceber que a utilização de anaglifos pode diminuir o custo de construção e utilização dessas ferramentas.

Além da avaliação com usuários da área de saúde, a continuidade deste trabalho prevê a avaliação do sistema com usuários diversos, variando-se os parâmetros que aqui foram

testados empiricamente. Pretende-se, com isso, confirmar os parâmetros escolhidos ou escolher parâmetros mais apropriados. Outra linha prevista é a avaliação da ferramenta com usuários que apresentam deficiências visuais, como astigmatismo e miopia, a fim de descobrir a influência de tais fenômenos na visualização estereoscópica.

O emprego de tecnologias gratuitas na implementação diminui os custos de construção e a escolha por anaglifos permite diminuir o custo de utilização, uma vez que é possível construir simples óculos de papel com lentes coloridas, que apresentam também custo muito baixo e fornecem a sensação de imersão através da observação de imagens anaglíficas. Os próximos passos do projeto serão justamente a constituição do pacote anteriormente mencionado para poder ser utilizado futuramente em outras aplicações do gênero e a implementação de outros métodos de baixo custo para proporcionar estereoscopia no sistema, como o *CromaDepth*, o que viria a sofisticar as ferramentas, possibilitando ao usuário escolher sob qual método visualizar um modelo 3D.

REFERÊNCIAS

ALVES, A. R. **Princípios Fundamentais da Estereoscopia**. UFSV – Santa Catarina,1999. Disponível em: <http://www.inf.ufsc.br/~visao/1999/aline/estereo.html>. Acesso em 18 abril 2005.

AUTODESK, Inc. ***3D Studio Max Documentation***. Disponível em: <http://www4.discreet.com/3dsmax/documentation>. Acesso em Agosto de 2005

FONTOURA, F.N.F. **Estereoscopia. Curso de Especialização em Informática com Ênfase em Internet e aplicações de ensino**. Disponível em < http://atlas.ucpel.tche.br/~magic/compgraf/estereoscopia.html >. Acessado em Julho de 2005.

HALUCK, R. *et al*. ***A Haptic Surgical Suturing Simulator***. Disponível em: <http://cs.millers,edu/haptics/suture.htm>. Acessado em julho de 2005.

HUNTER, P *et al*. ***Atlas of Clinical Ophthalmology***. Editora *Mosby*, vol.76, p 564-764, 1993.

IRELAND,P. ***Visualization of three dimensions datasets***. *Solar Physics*, vol.181, p. 87-90, *Kluwer Academic Publishers*, julho de 1998.

JOHANSON, M. ***Stereoscopic Video Transmission over the Internet***; Anais do WIAPP'01, *San Jose*, CA, Estados Unidos. Julho de 2001. Disponível em: <http://w2.alkit.se/~mathias/doc/wiapp-paper.pdf>. Acessado em Julho de 2005.

LIPTON, L. ***Foundations of the Stereoscopic Cinema.*** *Van Nostrand Reinhold Co*. p.77-79. 1982. Disponível em <www.mnemocine.com.br/ fotografia/estereo.htm>. Acesso em Julho de 2005.

MACHADO, L.S. *et al*. **Modelagem Tátil, Visualização Estereocópica e Aspectos de Avaliação em um Simulador de Coleta de Medula Óssea**, Anais do 3º Workshop Brasileiro de Realidade Virtual, 2000.

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 de São Paulo, São Paulo, 2003.

MACHADO, L. S. **A Realidade Virtual em Aplicações Científicas**. 1997. Grau: Dissertação (Mestrado em Computação Aplicada) - INPE, São José dos Campos, 1997.

MAHONEY, D.P. ***The Eyes Have it***. *Computer GraphicsWorld*, v. 21, n. 8, p. 69-70, 1998.

MANSSOUR, I.H. **Introdução à Java 3D**. 2003. Disponível em <www.inf.pucrs.br/manssour/Java3D>**.** Acesso em Julho de 2005.

MENESES, M.S. *et al*. **Estereoscopia aplicada à Neuroanatomia: estudo comparativo entre técnicas de filtro de cores e de polarização**. *Directory of Open Access Journals,* vol.60, p.769-774, 2002.

MONTANHA, F. **Aplicação de Realidade Virtual para construção de Atlas de Anatomia e Fisiopatologia do Câncer de Mama**. 2005. Grau: Dissertação (Mestrado em Ciência da Computação) - UNIVEM - Centro Universitário Eurípides de Marília - SP, 2005.

MORIE, J. F. ***Inspiring the Future: Merging Mass Communication, Art, Entertainment and Virtual Environments***, *Computer Graphics*, vol.28, p.135-138, 1994

PAIVA, J. *et al*. **Estereoscopia no Ensino da Química**.Química e Ensino, vol.86, p. 63-69, 2004.

PAVARINI, L *et al*. **Proposta de Implementação de Deformação em Ferramentas Virtuais de treinamento médico**. Anais do II Simpósio de Instrumentação e Imagens Médicas, vol.II, 2005.

SANTOS, E.T. **Uma Proposta para uso de Sistemas Estereoscópicos Modernos no Ensino de Geometria Descritiva e Desenho Técnico**, *Graphica* 2000, Ouro Preto(SP), p. 1-8, 2004. <Disponível em: http://docentes.pcc.usp.br/toledo/pdf/graphica2000_estereo.pdf>.

SCHWARTZ, J. ***Special topics in CS: intro to Bioinformatics***, 2004. Disponível em <http://www.settheory.com/bioinformatics_syllabus/bioinformatics_syllabus.html>. Acesso em Julho de 2005

SELMAN, D. ***Java 3D Programming***. Editora *Manning Publications Company*, 2002.

SISCOUTTO, R.A *et al*. **Realidade Virtual: Conceitos e Tendências**. Livro do Pré-Simpósio SVR 2004. Cap. 11, p.179-201. Editora Mania de Livro, São Paulo, 2004.

SOURIN, A. *et al*. ***Virtual Orthopedic Surgery Training***. IEEE *Computer Graphics and Applications*, vol.20, n.3, p.6-9, 2000.

SUN MICROSYSTEMS, Inc. ***Java 3D 1.2 API Documentation***. Disponível em <http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/j3dapi/>. Acesso em Julho de 2005.

YAGEL, R. *et al*. ***Building a Virtual Environment for Endoscopic Sinus Surgery Simulation***. *Computers & Graphics*, vol. 20, n. 6, p. 813-823, 1996.

APÊNDICES

Apêndice A

```
public class trabalho extends Applet {


private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private BranchGroup pai= new BranchGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;

private ConfiguredUniverse u;
private BranchGroup scene;
loader_normal ln= new loader_normal();
public BranchGroup createSceneGraph(SimpleUniverse
su) {   //branchgroup visao normal

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);

objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);

objRoot.setCapability(BranchGroup.ALLOW_DETACH
);
objRoot.setCapability(BranchGroup.ALLOW_CHILDR
EN_READ);
objRoot.setCapability(BranchGroup.ALLOW_CHILDR
EN_WRITE);
objRoot.setCapability(BranchGroup.ALLOW_CHILDR
EN_EXTEND);

pai.setCapability(BranchGroup.ALLOW_DETACH);
pai.setCapability(BranchGroup.ALLOW_CHILDREN_R
EAD);
pai.setCapability(BranchGroup.ALLOW_CHILDREN_
WRITE);
pai.setCapability(BranchGroup.ALLOW_CHILDREN_E
XTEND);
objRoot.addChild(pai);
pai.addChild(objRotate);
grupo = ln.carregaGaleao();
objRotate.addChild(grupo);
Vector3f transEsfera = new Vector3f(0.0f, 0.0f, -2.0f);
T3D.setTranslation(transEsfera);
objRotate.setTransform(T3D);

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.0 ,0.0 ,-2.0));

objRotate.setTransform(T3D1);

Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

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

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

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

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


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

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

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

TransformGroup vpTrans = null;
```

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

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

objRoot.compile();
return objRoot;
}
public void frame() {
setLayout(new BorderLayout());
GraphicsConfiguration config =
ConfiguredUniverse.getPreferredConfiguration();

Canvas3D canvas3D = new Canvas3D(config);
add("Center", canvas3D);
u = new ConfiguredUniverse(canvas3D);

scene=createSceneGraph(u);
u.getViewingPlatform().setNominalViewingTransform();
u.addBranchGraph(scene);



JFrame window = new JFrame();
window.setTitle("Visão Normal");
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(450, 400);
window.setLocation(40,0);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
JPanel p3 = new JPanel(new FlowLayout());

//cria os botões
JButton button5 = new JButton(" Camera Frontal ");
button5.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
1","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_frontal camera1= new anaglifo_frontal();
camera1.frame_cima1();
}
});
p3.add(button5);


JButton button6 = new JButton(" Camera Traseira ");
button6.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
2","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_traseiro camera2 = new anaglifo_traseiro();
camera2.frame_cima2();


}
});

p3.add(button6);
JButton button7 = new JButton(" Camera Esquerda ");
button7.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
3","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_esquerda camera3 = new anaglifo_esquerda();
camera3.frame_cima3();
}
});
p3.add(button7);
JButton button8 = new JButton(" Camera Direita ");
button8.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
4","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_direita camera4 = new anaglifo_direita();
camera4.frame_cima4();
}
});
p3.add(button8);

JButton button9 = new JButton(" Camera Superior ");
button9.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
5","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_superior camera5 = new anaglifo_superior();
camera5.frame_cima5();
}
});
p3.add(button9);

JButton button10 = new JButton(" Camera Inferior ");
button10.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
6","Atenção",JOptionPane.INFORMATION_MESSAGE
);

//            window.hide();
anaglifo_inferior camera6 = new anaglifo_inferior();
camera6.frame_cima6();
}
});
p3.add(button10);



container.add(canvas3D);
```

```
container.add(p2,BorderLayout.SOUTH);
container.add(p3,BorderLayout.SOUTH);



JPanel p4 = new JPanel(new FlowLayout());

//cria os botões
JButton button1 = new JButton("Olho Esquerdo");
button1.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
button1ActionPerformed(evt);
}
});
p4.add(button1);


JButton button2 = new JButton("Olho Direito");
button2.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
button2ActionPerformed(evt);
}
});

p4.add(button2);
JButton button4 = new JButton("Sair");
button4.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
button4ActionPerformed(evt);
}
});
p4.add(button4);



container.add(canvas3D);
container.add(p4,BorderLayout.NORTH);


window.show();

}


public void button1ActionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Carregando
Visao do Olho
Esquerdo","Atenção",JOptionPane.INFORMATION_ME
SSAGE);

visao_esquerda ve=new visao_esquerda();
ve.olhoesquerdo();

}


public void button2ActionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Carregando
Visao do Olho
Direito","Atenção",JOptionPane.INFORMATION_MES
SAGE);

visao_direita vd=new visao_direita();
vd.olhodireito();

}

public void button3ActionPerformed(ActionEvent evt) {
// Aqui pode coloca o que o botão faz

JOptionPane.showMessageDialog(null,"Carregando
Anaglifo","Atenção",JOptionPane.INFORMATION_ME
SSAGE);
anaglifo_frontal ana_f=new anaglifo_frontal();
ana_f.frame_cima1();

}

public void button4ActionPerformed(ActionEvent evt) {
// Aqui pode coloca o que o botão faz
System.exit(0);

}

//

//

////////////////////////////////////////////////////////////////////////
public static void main(String[] args) {
// Frame frame = new MainFrame(new CarregaVrml(),
800, 600);
trabalho carrega=new trabalho();
carrega.frame();


}

}
```

Apêndice B

```
                                                                    }
public class loader_normal extends Applet {

public Group carregaGaleao() //loader normal
{
Scene mama = null;
ObjectFile objFileloader = new
ObjectFile(ObjectFile.RESIZE);


try
{
mama = objFileloader.load("ductos.obj");
//                          galeao = objFileloader.load(new
java.net.URL(getCodeBase().toString() +
"./galleon.obj"));
}
catch (Exception e)
{
mama = null;
System.err.println(e);
}


BranchGroup branchGroup = mama.getSceneGroup();
branchGroup.setCollidable(false);
branchGroup.setCapability(BranchGroup.ALLOW_DET
ACH);
TransformGroup[] tg=mama.getViewGroups();
Behavior[] bh=mama.getBehaviorNodes();

Shape3D shape = (Shape3D) branchGroup.getChild(0);
shape.setCapability(Shape3D.ALLOW_LOCAL_TO_V
WORLD_READ);
Appearance app =  shape.getAppearance();

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_TEXTURE_AT
TRIBUTES_READ);
app.setCapability(Appearance.ALLOW_TEXTURE_AT
TRIBUTES_WRITE);



TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST
,0.4F);
app.setTransparencyAttributes(ta);




return branchGroup;

}
```

Apêndice C

```
public class loader_vermelho extends Applet {


public Group carregaGaleaovermelho() //loader vermelho
{
Scene mama = null;
ObjectFile objFileloader = new
ObjectFile(ObjectFile.RESIZE);


try
{
mama = objFileloader.load("mulher.obj");
//                          galeao = objFileloader.load(new
java.net.URL(getCodeBase().toString() +
"./galleon.obj"));
}
catch (Exception e)
{
mama = null;
System.err.println(e);
}


BranchGroup branchGroup = mama.getSceneGroup();

branchGroup.setCollidable(false);
branchGroup.setCapability(BranchGroup.ALLOW_DET
ACH);
TransformGroup[] tg=mama.getViewGroups();
Behavior[] bh=mama.getBehaviorNodes();




Shape3D shape = (Shape3D) branchGroup.getChild(0);
shape.setCapability(Shape3D.ALLOW_LOCAL_TO_V
WORLD_READ);
Appearance app =  shape.getAppearance();



Material material= new Material();
material.setEmissiveColor(new Color3f(165.0f, 0.0f,
0.0f) );

app.setMaterial(material);

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_WRITE);
TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST
,0.4F);
app.setTransparencyAttributes(ta);

return branchGroup;

}

}
```

Apêndice D

```
public class loader_azul extends Applet {


public Group carregaGaleaoazul() //loader azul
{
Scene mama = null;
ObjectFile objFileloader = new
ObjectFile(ObjectFile.RESIZE);


try
{
mama = objFileloader.load("mulher.obj");
//                          galeao = objFileloader.load(new
java.net.URL(getCodeBase().toString() +
"./galleon.obj"));
//galeao.setFlags();
}
catch (Exception e)
{
mama = null;
System.err.println(e);
}


BranchGroup branchGroup = mama.getSceneGroup();

branchGroup.setCollidable(false);
branchGroup.setCapability(BranchGroup.ALLOW_DET
ACH);
TransformGroup[] tg=mama.getViewGroups();
Behavior[] bh=mama.getBehaviorNodes();



Shape3D shape = (Shape3D) branchGroup.getChild(0);
shape.setCapability(Shape3D.ALLOW_LOCAL_TO_V
WORLD_READ);
Appearance app =  shape.getAppearance();



Material material= new Material();
material.setEmissiveColor(new Color3f(0.0f, 0.0f,
255.0f) );

app.setMaterial(material);

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_TRANSPAREN
CY_ATTRIBUTES_WRITE);
TransparencyAttributes ta = new
TransparencyAttributes(TransparencyAttributes.NICEST
,0.4F);
app.setTransparencyAttributes(ta);


return branchGroup;

}

}
```

Apêndice E

```
public void olhodireito()
                        //frame do olho direito
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene = createSceneGraph3(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);


JFrame window = new JFrame();
window.setTitle("Visão do Olho Direito");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(450, 400);
window.setLocation(490,400);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
JButton botaoazul = new JButton("Camada Azul");
botaoazul.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
botaoazulActionPerformed(evt);
}
});
p2.add(botaoazul);
container.add(canvas3D);
container.add(p2,BorderLayout.NORTH);
window.show();

}




public void botaoazulActionPerformed(ActionEvent evt)
{

JOptionPane.showMessageDialog(null,"Carregando
Camada
Azul","Atenção",JOptionPane.INFORMATION_MESS
AGE);
visao_direita_azul az=new visao_direita_azul();
az.azul();


}


public BranchGroup createSceneGraph3(SimpleUniverse
su) {       //branchgroup visao olho direito


BranchGroup objRoot = new BranchGroup();

Transform3D T3D = new Transform3D();

TransformGroup objRotate = new TransformGroup();
TransformGroup objRotate1 = new TransformGroup();


objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);

objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);

objRoot.addChild(objRotate);
objRoot.addChild(objRotate1);
/*
// Cria Esfera a posiciona em algum lugar (X, Y, Z)
Color3f bgColor = new Color3f(1.0f, 1.0f, 0.0f);
Transform3D t3 = new Transform3D ();
t3.setTranslation (new Vector3d(0.0, 1.0, -3.0));
TransformGroup objTrans2 = new TransformGroup(t3);
objRoot.addChild(objTrans2);
Material m = new Material(bgColor, bgColor, bgColor,
bgColor, 10.0f);
Appearance a = new Appearance();
m.setLightingEnable(true);
a.setMaterial(m);
Sphere sph = new Sphere(0.3f,
Sphere.GENERATE_NORMALS, 100, a);
objTrans2.addChild(sph);
*/


//seta cor para esfera
Appearance app = new Appearance();
ColoringAttributes ca = new ColoringAttributes();
ca.setColor(1.0f, 0.0f, 0.0f);

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );

app.setColoringAttributes(ca);

Group grupo = ln.carregaGaleao();
objRotate.addChild(grupo);

Vector3f transEsfera = new Vector3f(0.1f, 0.0f, -2.0f);
T3D.setTranslation(transEsfera);
objRotate.setTransform(T3D);

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.1 ,0.0 ,-2.0));
T3D1.rotY(Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate.setTransform(T3D1);
```

```
Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

/*              Color3f bgColor = new Color3f(0.05f, 0.05f,
0.2f);
Background bgNode = new Background(bgColor);
bgNode.setApplicationBounds(bounds);
objRoot.addChild(bgNode);*/

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

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

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

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


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

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

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

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

vpTrans =
su.getViewingPlatform().getViewPlatformTransform();
T3D.setTranslation(translate);
vpTrans.setTransform(T3D);

KeyNavigatorBehavior keyNavBeh = new
KeyNavigatorBehavior(vpTrans);
keyNavBeh.setSchedulingBounds(new
BoundingSphere(new Point3d(),1000.0));
objRoot.addChild(keyNavBeh);

objRoot.compile();
return objRoot;
}



}
```

Apêndice F

```
public void azul()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene = createSceneGraphAzul(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);


JFrame window = new JFrame();
window.setTitle("Visão do Olho Direito");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(450, 400);
window.setLocation(490,400);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());



container.add(canvas3D);

window.show();


}



public BranchGroup
createSceneGraphAzul(SimpleUniverse su) {
        //branchgroup azul


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

TransformGroup objRotate = new TransformGroup();
TransformGroup objRotate1 = new TransformGroup();


objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);

objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);

objRoot.addChild(objRotate);
objRoot.addChild(objRotate1);
/*
// Cria Esfera a posiciona em algum lugar (X, Y, Z)
Color3f bgColor = new Color3f(1.0f, 1.0f, 0.0f);
Transform3D t3 = new Transform3D ();
t3.setTranslation (new Vector3d(0.0, 1.0, -3.0));
TransformGroup objTrans2 = new TransformGroup(t3);
objRoot.addChild(objTrans2);
Material m = new Material(bgColor, bgColor, bgColor,
bgColor, 10.0f);
Appearance a = new Appearance();
m.setLightingEnable(true);
a.setMaterial(m);
Sphere sph = new Sphere(0.3f,
Sphere.GENERATE_NORMALS, 100, a);
objTrans2.addChild(sph);
*/


//seta cor para esfera
Appearance app = new Appearance();
ColoringAttributes ca = new ColoringAttributes();
ca.setColor(1.0f, 0.0f, 0.0f);

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );

app.setColoringAttributes(ca);

Group grupo = ln.carregaGaleao();
objRotate.addChild(grupo);
//              objRotate1.addChild(createLand());

Vector3f transEsfera = new Vector3f(0.1f, 0.0f, -2.0f);
T3D.setTranslation(transEsfera);
objRotate.setTransform(T3D);

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.1 ,10.0 ,-2.0));
T3D1.rotY(Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate.setTransform(T3D1);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

/*              Color3f bgColor = new Color3f(0.05f, 0.05f,
0.2f);
Background bgNode = new Background(bgColor);
bgNode.setApplicationBounds(bounds);
objRoot.addChild(bgNode);*/

// Inserindo Luz ambiente
Color3f ambientColor = new Color3f(0.0f, 0.0f, 1.0f);
```

```
AmbientLight ambientLightNode = new
AmbientLight(ambientColor);
ambientLightNode.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.0f, 0.0f, 1.0f);
Vector3f light1Direction  = new Vector3f(1.0f, 1.0f,
1.0f);
Color3f light2Color = new Color3f(0.0f, 0.0f, 1.0f);
Vector3f light2Direction  = new Vector3f(-1.0f, -1.0f, -
1.0f);

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

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


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

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

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

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}



}
```

Apêndice G

```
public class visao_esquerda extends Applet {

loader_normal ln= new loader_normal();


private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private BranchGroup pai= new BranchGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;



public void olhoesquerdo()                  //frame do olho
esquerdo
{

setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene = createSceneGraph2(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);


JFrame window = new JFrame();
window.setTitle("Visão do Olho Esquerdo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(450, 400);
window.setLocation(40,400);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
JButton botaovermelho = new JButton("Camada
Vermelha");
botaovermelho.addActionListener(new ActionListener()
{
public void actionPerformed(ActionEvent evt) {
botaovermelhoActionPerformed(evt);
}
});
p2.add(botaovermelho);
container.add(canvas3D);
container.add(p2,BorderLayout.NORTH);
window.show();


}

public void botaovermelhoActionPerformed(ActionEvent
evt) {

JOptionPane.showMessageDialog(null,"Carregando
Camada
Vermelha","Atenção",JOptionPane.INFORMATION_M
ESSAGE);
visao_esquerda_vermelha verm=new
visao_esquerda_vermelha();
verm.vermelho();



}

public BranchGroup createSceneGraph2(SimpleUniverse
su) {       //branchgroup visao olho esquerdo


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

TransformGroup objRotate = new TransformGroup();
TransformGroup objRotate1 = new TransformGroup();


objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);

objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);

objRoot.addChild(objRotate);
objRoot.addChild(objRotate1);
/*
// Cria Esfera a posiciona em algum lugar (X, Y, Z)
Color3f bgColor = new Color3f(1.0f, 1.0f, 0.0f);
Transform3D t3 = new Transform3D ();
t3.setTranslation (new Vector3d(0.0, 1.0, -3.0));
TransformGroup objTrans2 = new TransformGroup(t3);
objRoot.addChild(objTrans2);
Material m = new Material(bgColor, bgColor, bgColor,
bgColor, 10.0f);
Appearance a = new Appearance();
m.setLightingEnable(true);
a.setMaterial(m);
Sphere sph = new Sphere(0.3f,
Sphere.GENERATE_NORMALS, 100, a);
objTrans2.addChild(sph);

*/

//seta cor para esfera
Appearance app = new Appearance();
ColoringAttributes ca = new ColoringAttributes();
ca.setColor(1.0f, 0.0f, 0.0f);
```

```
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );

app.setColoringAttributes(ca);

Group grupo = ln.carregaGaleao();
objRotate.addChild(grupo);
//            objRotate1.addChild(createLand());

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

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.0 ,0.0 ,-2.0));
T3D1.rotY(-Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate.setTransform(T3D1);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

/*            Color3f bgColor = new Color3f(0.05f, 0.05f,
0.2f);
Background bgNode = new Background(bgColor);
bgNode.setApplicationBounds(bounds);
objRoot.addChild(bgNode);*/

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

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

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

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


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

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

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

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}



}
```

Apêndice H

```
public void vermelho()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphVermelho(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);


JFrame window = new JFrame();
window.setTitle("Visão do Olho Esquerdo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(450, 400);
window.setLocation(40,400);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
container.add(canvas3D);
window.show();

}



public BranchGroup
createSceneGraphVermelho(SimpleUniverse su) {
        //branchgroup vermelho


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

TransformGroup objRotate = new TransformGroup();
TransformGroup objRotate1 = new TransformGroup();


objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);

objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);

objRoot.addChild(objRotate);
objRoot.addChild(objRotate1);
/*
// Cria Esfera a posiciona em algum lugar (X, Y, Z)
Color3f bgColor = new Color3f(1.0f, 1.0f, 0.0f);
Transform3D t3 = new Transform3D ();
t3.setTranslation (new Vector3d(0.0, 1.0, -3.0));
TransformGroup objTrans2 = new TransformGroup(t3);
objRoot.addChild(objTrans2);
Material m = new Material(bgColor, bgColor, bgColor,
bgColor, 10.0f);
Appearance a = new Appearance();
m.setLightingEnable(true);
a.setMaterial(m);
Sphere sph = new Sphere(0.3f,
Sphere.GENERATE_NORMALS, 100, a);
objTrans2.addChild(sph);

*/

//seta cor para esfera
Appearance app = new Appearance();
ColoringAttributes ca = new ColoringAttributes();
ca.setColor(1.0f, 0.0f, 0.0f);

app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_WRITE);
app.setCapability(Appearance.ALLOW_COLORING_A
TTRIBUTES_READ);
app.setCapability(Appearance.ALLOW_MATERIAL_R
EAD );
app.setCapability(Appearance.ALLOW_MATERIAL_W
RITE );

app.setColoringAttributes(ca);

Group grupo = ln.carregaGaleao();
objRotate.addChild(grupo);
//          objRotate1.addChild(createLand());

Vector3f transEsfera = new Vector3f(0.1f, 0.0f, -2.0f);
T3D.setTranslation(transEsfera);
objRotate.setTransform(T3D);

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(-0.1 ,0.0 ,-2.0));
T3D1.rotY(-Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate.setTransform(T3D1);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

/*          Color3f bgColor = new Color3f(0.05f, 0.05f,
0.2f);
Background bgNode = new Background(bgColor);
bgNode.setApplicationBounds(bounds);
objRoot.addChild(bgNode);*/

// Inserindo Luz ambiente
Color3f ambientColor = new Color3f(1.0f, 0.0f, 0.0f);
AmbientLight ambientLightNode = new
AmbientLight(ambientColor);
ambientLightNode.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode);

// Inserindo Luz direcional
```

```
Color3f light1Color = new Color3f(1.0f, 0.0f, 0.0f);
Vector3f light1Direction  = new Vector3f(1.0f, 1.0f,
1.0f);
Color3f light2Color = new Color3f(1.0f, 0.0f, 0.0f);
Vector3f light2Direction  = new Vector3f(-1.0f, -1.0f, -
1.0f);

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

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


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

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

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

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}


}
```

Apêndice I

```
public class anaglifo_frontal extends Applet {

//globais
loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();
loader_normal ln = new loader_normal();

private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
//private Group grupo;


//fim globais



public void frame_cima1()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);


final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima1(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);


final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p = new JPanel(new FlowLayout());

JLabel label1= new JLabel();
label1.setIcon(new javax.swing.ImageIcon("1.jpg"));
p.add(label1);

JLabel label0= new JLabel("Olho Esquerdo = Vermelho
&");
p.add(label0);

JLabel label2= new JLabel("Olho Direito = Azul");
p.add(label2);

container.add(canvas3D);
container.add(p,BorderLayout.NORTH);
//          container.add(p5,BorderLayout.EAST);

window.show();

}



public BranchGroup
createSceneGraphCima1(SimpleUniverse su) {
          //branchgroup camera de frente


objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
//
          grupo.setCapability(Group.ALLOW_CHILDR
EN_WRITE);
//
          grupo.setCapability(Group.ALLOW_CHILDR
EN_READ);

objRotate.setCapability(TransformGroup.ALLOW_CHI
LDREN_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_CHI
LDREN_READ);
objRotate2.setCapability(TransformGroup.ALLOW_CHI
LDREN_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_CHI
LDREN_READ);

objRoot.addChild(objRotate);
objRoot.addChild(objRotate2);
//  pai.addChild(objRotate);
//  pai.addChild(objRotate2);



int a=0;
int b=0;
Group grupo2=  la.carregaGaleaoazul();
```

```
for (int i=0;i<=999999999;i++)
{
a=a+1;
b=b+1;
}

Group grupo =  lv.carregaGaleaovermelho();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

Transform3D T3D1 = new Transform3D();

T3D1.rotY(Math.PI/60);
T3D1.setTranslation( new Vector3d(0.005,-0.5 ,0.0));
objRotate.setTransform(T3D1);

//todo
/*                      Transform3D a = new
Transform3D();

a.setTranslation( new Vector3d(0.0 ,-1.0 ,0.0));
// a.rotX(Math.PI/2);
pai.setTransform(a);
*/


//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//     T3D2.setTranslation(Esfera);
//      objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
//            teste.rotY(-Math.PI/1);
teste.setTranslation (new Vector3d(-0.005,-0.5 ,0.0));
//  T3D1.rotY(Math.PI/12);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();


// Inserindo Luz ambiente
//Color3f ambientColor = new Color3f(1.0f,1.0f, 1.0f);
//            AmbientLight ambientLightNode = new
AmbientLight(ambientColor);
//
            ambientLightNode.setInfluencingBounds(boun
ds);
//            objRoot.addChild(ambientLightNode);

// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);

// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

DirectionalLight light2 = new
DirectionalLight(light2Color, light2Direction);
light2.setInfluencingBounds(bounds);
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                      light3.setInfluencingBounds(bounds);
//                      pai.addChild(light3);

//                      DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
//                      light4.setInfluencingBounds(bounds);
//               pai.addChild(light4);

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

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);

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

MouseTranslate myMouseTranslate2 = new
MouseTranslate();
myMouseTranslate2.setTransformGroup(objRotate2);
```

```
myMouseTranslate2.setSchedulingBounds(new
BoundingSphere());
objRotate2.addChild(myMouseTranslate2);


//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}

public static void main(String[] args) {
// Frame frame = new MainFrame(new CarregaVrml(),
800, 600);
anaglifo_frontal carrega=new anaglifo_frontal();
carrega.frame_cima1();


}



}
```

Apêndice J

```
public class anaglifo_traseiro extends Applet {

loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();

private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;



public void frame_cima2()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima2(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);

final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p = new JPanel(new FlowLayout());

JLabel label1= new JLabel();
label1.setIcon(new javax.swing.ImageIcon("1.jpg"));
p.add(label1);


JLabel label0= new JLabel("Olho Esquerdo = Vermelho
&");
p.add(label0);

JLabel label2= new JLabel("Olho Direito = Azul");
p.add(label2);

container.add(canvas3D);
container.add(p,BorderLayout.NORTH);

window.show();

}

public BranchGroup
createSceneGraphCima2(SimpleUniverse su) {
        //branchgroup camera pra cima

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
//objRoot.addChild(objRotate);
objRoot.addChild(pai);
pai.addChild(objRotate);
pai.addChild(objRotate2);

int a=0;
int b=0;
Group grupo =  la.carregaGaleaoazul();
for (int i=0;i<=999999999;i++)
{
a=a+1;
b=b+1;
}

Group grupo2      =  lv.carregaGaleaovermelho();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

//todo
Transform3D geral = new Transform3D();
geral.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
geral.rotY(-Math.PI);

pai.setTransform(geral);

Transform3D T3D1 = new Transform3D();
T3D1.rotY(Math.PI/60);
T3D1.setTranslation( new Vector3d(0.005,-0.5 ,0.0));

objRotate.setTransform(T3D1);
```

```
//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//   T3D2.setTranslation(Esfera);
//   objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
//            teste.rotY(-Math.PI/1);
teste.setTranslation (new Vector3d(-0.005,-0.5 ,0.0));
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);
// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

DirectionalLight light2 = new
DirectionalLight(light2Color, light2Direction);
light2.setInfluencingBounds(bounds);
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                        light3.setInfluencingBounds(bounds);
//                        pai.addChild(light3);

//                        DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
//                        light4.setInfluencingBounds(bounds);
//                pai.addChild(light4);
/*
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(objRotate);
myMouseRotate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate);

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);
*/
MouseTranslate myMouseTranslate = new
MouseTranslate();
myMouseTranslate.setTransformGroup(pai);
myMouseTranslate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseTranslate);

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}



}
```

Apêndice K

```
public class anaglifo_esquerda extends Applet {

loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();

private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;



public void frame_cima3()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima3(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);

final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p = new JPanel(new FlowLayout());

JLabel label1= new JLabel();
label1.setIcon(new javax.swing.ImageIcon("1.jpg"));
p.add(label1);


JLabel label0= new JLabel("Olho Esquerdo = Vermelho
&");
p.add(label0);

JLabel label2= new JLabel("Olho Direito = Azul");
p.add(label2);

container.add(canvas3D);
container.add(p,BorderLayout.NORTH);

window.show();

}

public BranchGroup
createSceneGraphCima3(SimpleUniverse su) {
        //branchgroup camera pra cima

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
//objRoot.addChild(objRotate);
objRoot.addChild(pai);
pai.addChild(objRotate);
pai.addChild(objRotate2);

int a=0;
int b=0;
Group grupo2                    =  la.carregaGaleaoazul();
for (int i=0;i<=999999999;i++)
{
a=a+1;
b=b+1;
}

Group grupo =  lv.carregaGaleaovermelho();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

//todo
Transform3D geral = new Transform3D();
geral.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
geral.rotY(-Math.PI/2);

pai.setTransform(geral);

Transform3D T3D1 = new Transform3D();
T3D1.rotY(Math.PI/60);
T3D1.setTranslation( new Vector3d(0.005,-0.5 ,0.0));

objRotate.setTransform(T3D1);
```

```
//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//   T3D2.setTranslation(Esfera);
//   objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
//            teste.rotY(-Math.PI/1);
teste.setTranslation (new Vector3d(-0.005,-0.5 ,0.0));
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);

Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);
// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

DirectionalLight light2 = new
DirectionalLight(light2Color, light2Direction);
light2.setInfluencingBounds(bounds);
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                        light3.setInfluencingBounds(bounds);
//                        pai.addChild(light3);

//                        DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
//                        light4.setInfluencingBounds(bounds);
//                pai.addChild(light4);
/*
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(objRotate);
myMouseRotate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate);

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);
*/
MouseTranslate myMouseTranslate = new
MouseTranslate();
myMouseTranslate.setTransformGroup(pai);
myMouseTranslate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseTranslate);

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}

}
```

Apêndice L

```
public class anaglifo_direita extends Applet {

loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();

private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;



public void frame_cima4()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima4(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);

final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p = new JPanel(new FlowLayout());

JLabel label1= new JLabel();
label1.setIcon(new javax.swing.ImageIcon("1.jpg"));
p.add(label1);


JLabel label0= new JLabel("Olho Esquerdo = Vermelho
&");
p.add(label0);

JLabel label2= new JLabel("Olho Direito = Azul");
p.add(label2);

container.add(canvas3D);
container.add(p,BorderLayout.NORTH);

window.show();

}

public BranchGroup
createSceneGraphCima4(SimpleUniverse su) {
        //branchgroup camera pra cima

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
//objRoot.addChild(objRotate);
objRoot.addChild(pai);
pai.addChild(objRotate);
pai.addChild(objRotate2);

int a=0;
int b=0;
Group grupo                    =  la.carregaGaleaoazul();
for (int i=0;i<=999999999;i++)
{
a=a+1;
b=b+1;
}

Group grupo2 =  lv.carregaGaleaovermelho();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

//todo
Transform3D geral = new Transform3D();
geral.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
geral.rotY(Math.PI/2);

pai.setTransform(geral);

Transform3D T3D1 = new Transform3D();
T3D1.rotY(Math.PI/60);
T3D1.setTranslation( new Vector3d(0.005,-0.5 ,0.0));

objRotate.setTransform(T3D1);
```

```
//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//   T3D2.setTranslation(Esfera);
//   objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
//            teste.rotY(-Math.PI/1);
teste.setTranslation (new Vector3d(-0.005,-0.5 ,0.0));
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);

Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);
// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

DirectionalLight light2 = new
DirectionalLight(light2Color, light2Direction);
light2.setInfluencingBounds(bounds);
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                        light3.setInfluencingBounds(bounds);
//                        pai.addChild(light3);

//                        DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
//                        light4.setInfluencingBounds(bounds);
//                pai.addChild(light4);
/*
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(objRotate);
myMouseRotate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate);

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);
*/
MouseTranslate myMouseTranslate = new
MouseTranslate();
myMouseTranslate.setTransformGroup(pai);
myMouseTranslate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseTranslate);

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}

}
```

Apêndice M

```
public class anaglifo_superior extends Applet {

loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();

private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;

public void frame_cima5()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima5(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);

final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
JPanel p3 = new JPanel(new FlowLayout());

//cria os botões
JButton button5 = new JButton(" Camera Frontal ");
button5.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
1","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_frontal camera1= new anaglifo_frontal();
camera1.frame_cima1();
}
});
p2.add(button5);


JButton button6 = new JButton(" Camera Traseira ");
button6.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
2","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_traseiro camera2 = new anaglifo_traseiro();
camera2.frame_cima2();


}
});

p2.add(button6);
JButton button7 = new JButton(" Camera Esquerda ");
button7.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
3","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_esquerda camera3 = new anaglifo_esquerda();
camera3.frame_cima3();
}
});
p2.add(button7);
JButton button8 = new JButton(" Camera Direita ");
button8.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
4","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_direita camera4 = new anaglifo_direita();
camera4.frame_cima4();
}
});
p3.add(button8);

JButton button9 = new JButton(" Camera Superior ");
button9.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
5","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_superior camera5 = new anaglifo_superior();
camera5.frame_cima5();
}
});
p3.add(button9);

JButton button10 = new JButton(" Camera Inferior ");
button10.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
```

```
JOptionPane.showMessageDialog(null,"Camera
6","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_inferior camera6 = new anaglifo_inferior();
camera6.frame_cima6();
}
});
p3.add(button10);       container.add(canvas3D);
container.add(p2,BorderLayout.NORTH);
container.add(p3,BorderLayout.SOUTH);
//            container.add(p4,BorderLayout.WEST);
//            container.add(p5,BorderLayout.EAST);


window.show();

}


public BranchGroup
createSceneGraphCima5(SimpleUniverse su) {
            //branchgroup camera pra cima

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
objRoot.addChild(pai);
pai.addChild(objRotate);
pai.addChild(objRotate2);

Group grupo = lv.carregaGaleaovermelho();
Group grupo2= la.carregaGaleaoazul();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

//todo
Transform3D a = new Transform3D();
a.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
a.rotX(-Math.PI/2);
pai.setTransform(a);

Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
T3D1.rotY(Math.PI/50);
objRotate.setTransform(T3D1);

//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//     T3D2.setTranslation(Esfera);
//      objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
teste.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
teste.rotY(-Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();

// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);
// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

DirectionalLight light2 = new
DirectionalLight(light2Color, light2Direction);
light2.setInfluencingBounds(bounds);
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                      light3.setInfluencingBounds(bounds);
//                      pai.addChild(light3);

//                      DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
```

```
//                      light4.setInfluencingBounds(bounds);
//              pai.addChild(light4);
/*
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(objRotate);
myMouseRotate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate);

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);
*/
MouseTranslate myMouseTranslate = new
MouseTranslate();
myMouseTranslate.setTransformGroup(pai);
myMouseTranslate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseTranslate);

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}



}
```

Apêndice N

```
public class anaglifo_inferior extends Applet {

loader_vermelho lv= new loader_vermelho();
loader_azul la= new loader_azul();


private BranchGroup paiRoot = new BranchGroup();
private BranchGroup objRoot = new BranchGroup();
private Transform3D T3D = new Transform3D();
private Transform3D transpai = new Transform3D();
private Transform3D T3D2 = new Transform3D();
private Transform3D testet=new Transform3D();
private TransformGroup pai= new TransformGroup();
private TransformGroup objRotate = new
TransformGroup();
private TransformGroup objRotate1 = new
TransformGroup();
private TransformGroup objRotate2 = new
TransformGroup();
private ObjectFile loader = new ObjectFile();
private Group grupo;


public void frame_cima6()
{
setLayout(new BorderLayout());
Canvas3D canvas3D = new Canvas3D(null);
add("Center", canvas3D);

final SimpleUniverse simpleU = new
SimpleUniverse(canvas3D);

BranchGroup scene =
createSceneGraphCima6(simpleU);

simpleU.getViewingPlatform().setNominalViewingTrans
form();
simpleU.addBranchGraph(scene);

final JFrame window = new JFrame();

window.setTitle("Anaglifo");
//
window.setDefaultCloseOperation(JFrame.EXIT_ON_C
LOSE);
window.setSize(650, 600);
window.setLocation(300,0);

Container container =window.getContentPane();

JPanel p2 = new JPanel(new FlowLayout());
JPanel p3 = new JPanel(new FlowLayout());

//cria os botões
JButton button5 = new JButton(" Camera Frontal ");
button5.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
1","Atenção",JOptionPane.INFORMATION_MESSAGE
);
window.hide();
anaglifo_frontal camera1= new anaglifo_frontal();
camera1.frame_cima1();
}
});
p2.add(button5);



JButton button6 = new JButton(" Camera Traseira ");
button6.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
2","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_traseiro camera2 = new anaglifo_traseiro();
camera2.frame_cima2();



}
});

p2.add(button6);
JButton button7 = new JButton(" Camera Esquerda ");
button7.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
3","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_esquerda camera3 = new anaglifo_esquerda();
camera3.frame_cima3();
}
});
p2.add(button7);
JButton button8 = new JButton(" Camera Direita ");
button8.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
4","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_direita camera4 = new anaglifo_direita();
camera4.frame_cima4();
}
});
p3.add(button8);

JButton button9 = new JButton(" Camera Superior ");
button9.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
5","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_superior camera5 = new anaglifo_superior();
camera5.frame_cima5();
}
});
p3.add(button9);

JButton button10 = new JButton(" Camera Inferior ");
```

```
button10.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent evt) {
JOptionPane.showMessageDialog(null,"Camera
6","Atenção",JOptionPane.INFORMATION_MESSAGE
);

window.hide();
anaglifo_inferior camera6 = new anaglifo_inferior();
camera6.frame_cima6();
}
});
p3.add(button10);




container.add(canvas3D);
container.add(p2,BorderLayout.NORTH);
container.add(p3,BorderLayout.SOUTH);
//            container.add(p4,BorderLayout.WEST);
//            container.add(p5,BorderLayout.EAST);


window.show();

}


public BranchGroup
createSceneGraphCima6(SimpleUniverse su) {
            //branchgroup camera pra cima

objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_WRITE);
objRotate.setCapability(TransformGroup.ALLOW_TRA
NSFORM_READ);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate1.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_WRITE);
objRotate2.setCapability(TransformGroup.ALLOW_TR
ANSFORM_READ);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_WRITE);
pai.setCapability(TransformGroup.ALLOW_TRANSFO
RM_READ);
pai.setCapability(1);
objRoot.addChild(pai);
pai.addChild(objRotate);
pai.addChild(objRotate2);

Group grupo = lv.carregaGaleaovermelho();
Group grupo2= la.carregaGaleaoazul();
objRotate.addChild(grupo);
objRotate2.addChild(grupo2);

Vector3f transEsferapai = new Vector3f(0.0f, 0.0f, -1.0f);
transpai.setTranslation(transEsferapai);
pai.setTransform(transpai);

//todo
Transform3D a = new Transform3D();
a.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
a.rotX(Math.PI/2);
pai.setTransform(a);


Transform3D T3D1 = new Transform3D();
T3D1.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
T3D1.rotY(Math.PI/50);

objRotate.setTransform(T3D1);


//            Vector3f Esfera = new Vector3f(0.1f, 0.0f, -
2.0f);
//    T3D2.setTranslation(Esfera);
//     objRotate2.setTransform(T3D2);

Transform3D teste = new Transform3D();
teste.setTranslation( new Vector3d(0.0 ,0.0 ,0.0));
teste.rotY(-Math.PI/50);
//  Transform3D T3D12 = new Transform3D();
//  T3D12.rotY(Math.PI/2);
// T3D1.mul(T3D12);
objRotate2.setTransform(teste);


Vector3f transLand = new Vector3f(0.0f, -0.45f, 0.0f);
T3D.setTranslation(transLand);
objRotate1.setTransform(T3D);

BoundingBox bounds = new BoundingBox();


// Inserindo Luz ambiente objRotate-vermelho
Color3f ambientColor2 = new Color3f(255.0f,0.0f, 0.0f);
AmbientLight ambientLightNode2 = new
AmbientLight(ambientColor2);
ambientLightNode2.setInfluencingBounds(bounds);
objRotate.addChild(ambientLightNode2);
// Inserindo Luz ambiente objRotate2-azul
Color3f ambientColor3 = new Color3f(0.0f,0.0f, 255.0f);
AmbientLight ambientLightNode3 = new
AmbientLight(ambientColor3);
ambientLightNode3.setInfluencingBounds(bounds);
objRotate2.addChild(ambientLightNode3);

// Inserindo Luz direcional
Color3f light1Color = new Color3f(0.2f, 0.2f,0.2f);
Vector3f light1Direction  = new Vector3f(0.01f, 0.0f,
0.0f);
Color3f light2Color = new Color3f(0.2f, 0.2f, 0.2f);
Vector3f light2Direction  = new Vector3f(-0.01f,0.0f,
0.0f);

//            Color3f light3Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light3Direction  = new Vector3f(0.0f,
0.01f, 0.0f);
//            Color3f light4Color = new Color3f(0.2f, 0.2f,
0.2f);
//            Vector3f light4Direction  = new Vector3f(0.0f,-
0.01f, 0.0f);

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

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

```
objRotate2.addChild(light2);

//              DirectionalLight light3 = new
DirectionalLight(light3Color, light3Direction);
//                      light3.setInfluencingBounds(bounds);
//                      pai.addChild(light3);

//                      DirectionalLight light4 = new
DirectionalLight(light4Color, light4Direction);
//                      light4.setInfluencingBounds(bounds);
//               pai.addChild(light4);
/*
MouseRotate myMouseRotate = new MouseRotate();
myMouseRotate.setTransformGroup(objRotate);
myMouseRotate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate);

MouseRotate myMouseRotate2 = new MouseRotate();
myMouseRotate2.setTransformGroup(objRotate2);
myMouseRotate2.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseRotate2);
*/
MouseTranslate myMouseTranslate = new
MouseTranslate();
myMouseTranslate.setTransformGroup(pai);
myMouseTranslate.setSchedulingBounds(new
BoundingSphere());
objRoot.addChild(myMouseTranslate);

//cria base e aciona o teclado
TransformGroup vpTrans = null;

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

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

objRoot.compile();
return objRoot;
}



}
```

Apêndice O

Dados pessoais do avaliador:

1. Idade:
[ ]Abaixo de 20 anos
[ ]20 a 30 anos
[ ]31 a 40 anos
[ ]41 a 50 anos
[ ]51 a 60 anos
[ ]acima de 60 anos

2. Escolaridade:
[ ]Cursando Graduação
[ ]Graduação completa
[ ]Especialização completa
[ ]Mestrado completo
[ ]Doutorado completo
[ ]Pós-Doutorado completo

3. Profissão:
[ ]Estudante
[ ]Professor/Pesquisador
[ ]Profissional da saúde
[ ]Outros

4. Área de Atuação:
[ ]Engenharia
[ ]Computação
[ ]Saúde
[ ]Física
[ ]Outros

5. Trabalha com sistemas de auxílio ao diagnóstico?
[ ]Não
[ ]Sim, com desenvolvimento
[ ]Sim, como usuário
[ ]Sim, como avaliador

Avaliação do sistema

1. Facilidade de entendimento do sistema:
[ ]Ótimo
[ ]Bom
[ ]Regular
[ ]Ruim

2. Facilidade de acesso às funções do sistema:
[ ]Ótimo
[ ]Bom
[ ]Regular
[ ]Ruim

3. Visualização de apenas um modelo estéreo:
[ ]Ótimo
[ ]Bom
[ ]Regular
[ ]Ruim

4. Identificação de apenas uma borda:
[ ]Ótimo
[ ]Bom
[ ]Regular
[ ]Ruim

5. Visualização do modelo estéreo em relevo:
[ ]Ótimo
[ ]Bom
[ ]Regular
[ ]Ruim

6. Considera este sistema apto para auxiliar o treinamento médico?
[ ]Sim
[ ]Não

7. Comentários