PAULO MUGGLER MOREIRA

Serviço de Bibliotecas
Biblioteca de Engenharia Elétrica

PROJEÇÕES ESTEREOSCÓPICAS PARA REALIDADE AUMENTADA ESPACIAL UTILIZANDO SHADERS NO JAVA3D

Projeto apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

São Paulo

2008

PAULO MUGGLER MOREIRA

IMPLEMENTAÇÃO DE PROJEÇÕES ESTEREOSCÓPICAS PARA REALIDADE AUMENTADA ESPACIAL UTILIZANDO SHADERS NO JAVA3D

Projeto apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Área de Concentração:
Engenharia da Computação – Computação Gráfica e Realidade Aumentada

Orientador: Professor Dr. Romero Tori

São Paulo

2008

Ao meu pai; sem o seu apoio e incentivo, este momento não seria possível. À minha mãe; pela orientação e o conforto oferecidos nos momentos mais difíceis.

AGRADECIMENTOS

Aos professores Dr. Romero Tori e Daniel Calife, orientadores que possibilitaram a realização deste projeto, e cujas opiniões ao longo do projeto foram valiosas.

Aos professores da Escola Politécnica e especialmente do PCS, que contribuíram para o meu desenvolvimento como Engenheiro.

Aos meus pais, que sempre fizeram de tudo para que eu chegasse nos meus objetivos.

RESUMO

Este trabalho contém os detalhes da implementação de um sistema que realiza projeções estereoscópicas de universos virtuais do Java 3D utilizando shaders programáveis. A finalidade deste sistema é a sua utilização em aplicações de Realidade Aumentada Espacial. O efeito de estereoscopia é obtido através de anaglifos. São apresentados os principais conceitos envolvidos na projeção de imagens estereoscópicas em ambientes com aplicações de Realidade Aumentada Espacial.

Palavras-chave: estereoscopia, anaglifos, shaders programáveis, Java 3D, OpenGL, Realidade Aumentada Espacial.

ABSTRACT

This work details the implementation of a system that achieves stereoscopic projections of Java 3D virtual universes, using programmable shaders. The purpose of this system is to provide a means of stereoscopic viewing for Spatial Augmented Reality applications. The stereoscopic effect described here is obtained through the use of anaglyphs. The concepts involved herein atre thoroughly described.

Keywords: stereoscopy, anaglyphs, programmable shaders, Java 3D, OpenGL, Spatial Augmented Reality..

LISTA DE FIGURAS

Figura 2-1: Espectro de Realidade Virtual 21

Figura 2-2: RA com vaso e carro virtuais sobre a mesa 22

Figura 2-3: Ferramentas de Interação para Aplicação de RA: plano de corte (abaixo) e apontador (acima). 24

Figura 2-4: O Painel de Interação Pessoal permite a interação com widgets 2D e 3D utilizando as duas mãos. 25

Figura 2-5: Componentes do projeto Pinchglove, em sentido horário: a) Capacete HMD de combinação óptica com câmera para rastreamento óptico e um rastreador inercial; b) Luvas e touch pad utilizados em conjunto; c) Painel touch pad aumentado com interface da aplicação; d) Componentes da Luva Pinchglove. 26

Figura 2-6: Imagem vista pelo observador com olhos fixos no objeto próximo 28

Figura 2-7: Imagem vista pelo observador com olhos fixos no objeto distante 29

Figura 2-8: Paralelepípedos em perspectiva 29

Figura 2-9: Círculo e Hexágono 30

Figura 2-10: Esfera e Cubo 30

Figura 2-11: Esfera atrás do Cubo 30

Figura 2-12: Esfera na frente do Cubo 30

Figura 2-13: Efeitos da sombra 31

Figura 2-14: Disparidade da retina 31

Figura 2-15: Paralaxe negativa 32

Figura 2-16: Paralaxe zero 32

Figura 2-17: Paralaxe positiva 32

Figura 2-18: Valores de paralaxe 33

Figura 2-19: Obtenção de um par estereoscópico – separação horizontal entre as câmeras 34

Figura 2-20: Esquema simplificado da técnica de lentes polarizadas 35

Figura 2-21: Óculos obturadores 36

Figura 2-22: Óculos utilizados para visão estéreo com anaglifo 37

Figura 2-23: Separação de cores através dos óculos anaglíficos 37

Figura 2-24: Exemplo de anaglifo 37

Figura 2-25: Anaglifo puro 39
Figura 2-26: Anaglifo em escala de cinza 40
Figura 2-27: Anaglifo em cores 41
Figura 2-28: Anaglifo semi-colorido 41
Figura 2-29: Anaglifo otimizado 42
Figura 2-30: O Pipeline gráfico do OpenGL - funcionalidade fixa 43
Figura 2-31: Resumo visual do processamento realizado pelo pipeline gráfico do OpenGL 45
Figura 2-32: Pipeline do OpenGL com funcionalidade fixa substituída por shaders programáveis 45
Figura 2-33: Processo de substituição da funcionalidade fixa em uma aplicação OpenGL 46
Figura 2-34: Organização em camadas de um sistema utilizando Java3D 47
Figura 2-35: Grafo de cena do Java3D. 49
Figura 2-36: Encapsulamento oferecido pela classe SimpleUniverse 51
Figura 2-42: Hierarquia de classes que implementa a funcionalidade de shaders programáveis no Java3D (diagrama simplificado) 54
Figura 4-1- Óculos utilizados para visão estéreo com anaglifo. 64
Figura 4-2: Diagrama de blocos para a produção de um anaglifo 66
Figura 4-3: Diagrama de classes do projeto principal 67
Figura 5-1: Grafo de cena contendo apenas um cubo 70
Figura 5-2: Renderização do objeto com um fragment shader simples associado 71
Figura 5-3: Aplicação de dois fragment shaders, um para a cor azul e outro para a cor vermelha do anaglifo 71
Figura 5-4: Renderização de um cubo com anaglifo 72
Figura 5-5: Renderização do anaglifo em diferentes modos: a) puro; b) escala de cinza; c) em cores; d) semi colorido; e) otimizado 73
Figura 5-6: Cena com separações interoculares diferentes: a) separação menor; b) separação maior 74
Figura 5-7: Renderização do grafo de cena com múltiplos objetos 75
Figura 5-8: Grafo de cena com múltiplos objetos animados e com interatividade 76
Figura 5-9: Interface gráfica do aplicativo 77
Figura 5-10: Classe AnaglyphStereoFrame: Menu File 82

Figura 5-11: Classe AnaglyphStereoFrame: Menu View 83
Figura 6-1: Amostra do efeito obtido 85
Figura 6-2: Picos de processamento do programa..88

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface

RA – Realidade Aumentada

RAE – Realidade Aumentada Espacial

SUMÁRIO

1. INTRODUÇÃO 14

1.1 DEFINIÇÃO DO PROBLEMA E MOTIVAÇÃO 14
1.2 OBJETIVOS DESTE TRABALHO 16
1.2.1 IMPLEMENTAÇÃO DE PROJEÇÕES ESTEREOSCÓPICAS EM AMBIENTE COM RAE UTILIZANDO SHADERS NO JAVA 3D 16
1.3 ESTRUTURA DESTE TRABALHO 17

2. ASPECTOS CONCEITUAIS 18

2.1 COMPUTAÇÃO GRÁFICA: HISTÓRIA E EVOLUÇÃO 18
2.2 REALIDADE AUMENTADA 20
2.2.1 REALIDADE AUMENTADA ESPACIAL 23
2.2.2 EXEMPLOS DE APLICAÇÕES DE RA: TÉCNICAS DE INTERAÇÃO 24
2.3 ESTEREOSCOPIA E VISÃO TRIDIMENSIONAL 27
2.3.1 VISÃO TRIDIMENSIONAL 27
2.4 TÉCNICAS DE ESTEREOSCOPIA 33
2.4.1 ÓCULOS POLARIZADOS 34
2.4.2 ÓCULOS OBTURADORES 35
2.4.3 ANAGLIFOS 36
2.5 O PIPELINE GRÁFICO DO OPENGL 42
2.6 A API JAVA 3D 47
2.6.1 GRAFOS DE CENA 48
2.6.2 CLASSES PRINCIPAIS DO JAVA3D 50
2.6.3 SHADERS PROGRAMÁVEIS NO JAVA 3D: HIERARQUIA DE CLASSES 53
2.6.4 SHADERS PROGRAMÁVEIS NO JAVA 3D: EXEMPLO DE FUNCIONAMENTO 57

3. METODOLOGIA 59

3.1 METODOLOGIA DE PESQUISA 59
3.2 METODOLOGIA DE PROJETO E DESENVOLVIMENTO 60

4. ESPECIFICAÇÃO DO PROJETO 63

4.1 REQUISITOS DE HARDWARE 63
4.2 REQUISITOS DE SOFTWARE 64
4.3 CONSIDERAÇÕES SOBRE OS USUÁRIOS DO SISTEMA 64
4.4 FUNCIONALIDADE IMPLEMENTADA 65
4.5 DIAGRAMA DE BLOCOS DO PROCESSO DE PRODUÇÃO DE UM ANAGLIFO 65
4.6 DIAGRAMA DE CLASSES DO SISTEMA 67

5. PROJETO E IMPLEMENTAÇÃO 70

5.1 DESCRIÇÃO DAS ITERAÇÕES DE DESENVOLVIMENTO 70
5.1.1 CONSTRUÇÃO DE UM GRAFO DE CENA SIMPLES NO JAVA 3D 70
5.1.2 CODIFICAÇÃO E APLICAÇÃO DE UM SHADER PROGRAMÁVEL SIMPLES 70
5.1.3 ALTERAÇÃO DO SHADER PARA A OBTENÇÃO DAS CORES DO ANAGLIFO 71
5.1.4 APLICAÇÃO DE DOIS SHADERS E SOMA DAS IMAGENS RESULTANTES 71
5.1.5 CONTROLE DO MODO DE PRODUÇÃO DO ANAGLIFO 73
5.1.6 CONTROLE DA SEPARAÇÃO INTEROCULAR DO OBSERVADOR 73
5.1.7 INTRODUÇÃO DE MÚLTIPLOS OBJETOS NO GRAFO DE CENA 74
5.1.8 SUPORTE À RENDERIZAÇÃO DE CENAS DINÂMICAS 75
5.1.9 IMPLEMENTAÇÃO DE UMA INTERFACE GRÁFICA SIMPLES 77
5.1.10 CRIAÇÃO DE UM FRAMEWORK E OTIMIZAÇÃO DO CÓDIGO GERADO 77
5.2 DESCRIÇÃO DAS CLASSES CONTIDAS NA VERSÃO FINAL DO SOFTWARE 78
5.2.1 CLASSE ANAGLYPHSTEREOUNIVERSE 78
5.2.2 CLASSE ANAGLYPHSHADERAPPEARANCE 80
5.2.3 CLASSE ANAGLYPHSHADERATTRIBUTESET 80
5.2.4 CLASSE BUFFEREDIMAGEPANEL 81
5.2.5 CLASSE CONTENTBRANCH 81
5.2.6 CLASSE APPLETSTEREOVIEW 82
5.2.7 CLASSE ANAGLYPHSTEREOFRAME 82
5.2.8 SHADER PROGRAMÁVEL 83

6. RESULTADOS 85

6.1 AMOSTRA DO EFEITO ESTEREOSCÓPICO 85

6.2 AVALIAÇÃO DO EFEITO ESTEREOSCÓPICO 86
6.3 DESEMPENHO DO SISTEMA EM TAXA DE QUADROS 86

7. CONCLUSÃO E TRABALHOS FUTUROS 89

7.1 CUMPRIMENTO DAS METAS PROPOSTAS 89
7.1.1 CRIAÇÃO DO EFEITO ESTEREOSCÓPICO 89
7.1.2 ANIMAÇÃO 89
7.1.3 INTERATIVIDADE 89
7.1.4 DESEMPENHO 89
7.2 LIMITAÇÕES DO SISTEMA E POSSÍVEIS MELHORIAS 90
7.2.1 DESEMPENHO 90
7.2.2 FUNCIONALIDADE 91
7.3 APLICABILIDADE EM AMBIENTES DE REALIDADE AUMENTADA ESPACIAL 92
7.4 INTEGRAÇÃO COM O ENJINE 92

8. BIBLIOGRAFIA 94

1. INTRODUÇÃO

Este capítulo contém uma breve exposição do contexto geral (tecnológico, social e econômico) em que este projeto se insere, bem como a motivação que levou ao seu desenvolvimento. São enunciados os objetivos do projeto, e uma breve descrição das seções contidas neste documento é apresentada.

1.1 DEFINIÇÃO DO PROBLEMA E MOTIVAÇÃO

A computação gráfica é o ramo da ciência da computação que lida com os conceitos e técnicas da síntese de imagens utilizando o computador (McGraw-Hill 2004). Entre 1947 e 1951, pesquisadores do *MIT* (*Massachussets Institute of Technology)* desenvolveram um projeto que é considerado por muitos o marco inicial do nascimento da computação gráfica, o projeto *Whirlwind*. (Redmond e Smith 1980)

O *Whirlwind* foi o primeiro computador digital capaz de operar em tempo real, utilizando um dispositivo de saída de vídeo como interface de saída. O desenvolvimento do *Whirlwind* teve como conseqüência direta a produção do *SAGE (Semi Automatic Ground Environment)*, o sistema de monitoramento de espaço aéreo da Força Aérea Americana na década de 60. Indiretamente, os avanços produzidos no campo da computação gráfica no projeto *Whirlwind* fizeram parte das condições que tornaram possível o conceito do computador pessoal (Redmond e Smith 1980).

A computação gráfica sofreu uma imensa evolução desde o seu surgimento até os dias atuais, criando hoje possibilidades que seriam inimagináveis há alguns anos. Se em seu surgimento ela era restrita às aplicações militares de alta importância estratégica, com a necessidade de complexos laboratórios de pesquisa e patrocínio direto do governo para seu desenvolvimento, hoje em dia temos que todos os setores da sociedade se utilizam dos recursos da computação gráfica direta ou indiretamente. Desde as aplicações médicas de diagnóstico por imagens geradas em computador, passando pelas aplicações de *CAD (Computer Aided Design)* em engenharia, até os prosaicos jogos eletrônicos, os avanços da tecnologia da informação criam a todo instante novas possibilidades e demandas cada vez mais complexas no campo da computação gráfica.

Neste contexto encontra-se a Realidade Aumentada (RA), campo de pesquisa da computação gráfica cujo objetivo é proporcionar a visualização de elementos virtuais em sobreposição ao mundo físico, permitindo também a interação com estes elementos em tempo real (Raskar, Spatially Augmented Reality 1998). Compreendendo a que se propõe este campo de pesquisa relativamente novo que é a RA, podemos vislumbrar uma revolução de proporções ainda maiores do que aquela iniciada pelo projeto *Whirlwind* na década de 50. Cenas que hoje em dia têm lugar nos filmes de ficção científica mais futuristas podem bem se tornar realidade em algumas décadas, dependendo dos avanços realizados no campo da RA nos próximos anos. Por exemplo: imagine-se visitando o museu do Louvre, localizado em Paris, na França. No entanto, não estamos em Paris, mas sim em São Paulo, ou talvez no Rio de Janeiro, ou em outra cidade qualquer. Você pode ver todas as obras de arte com perfeição; Da Vinci, Van Gogh, Rafael, estão todos lá, mas na verdade tudo não passa de um truque de sofisticação tecnológica concebido para nos enganar os sentidos. Será apenas uma fantasia distante? Pode bem ser, da mesma forma que 50 anos atrás os avanços tecnológicos dos quais nos valemos corriqueiramente em nosso cotidiano não faziam parte nem mesmo das obras de ficção.

O campo da RA possui de fato o potencial para revolucionar novamente as atividades humanas da mesma forma que a computação gráfica o fez nos últimos 50 anos. Entretanto, ainda não estamos muito além de onde se encontravam os pesquisadores do MIT ao desenvolver o *Whirlwind*. Existem inúmeras barreiras de naturezas diversas a serem transpostas, exigindo que se façam pesquisas intensas em diversas áreas e que sejam desenvolvidas ferramentas e soluções que se encontram ainda na fase embrionária de sua evolução.

Essa carência é o principal motivador para este projeto, que busca alternativas às técnicas de estereoscopia existentes para aplicações de Realidade Aumentada Espacial (RAE). A grande maioria das técnicas de estereoscopia em uso nas aplicações de RA atuais exigem dispositivos especiais de visualização, como: óculos com obturadores, lentes polarizadoras de luz, ou HMDs *(Head Mounted Displays)*. Estes métodos possuem algumas desvantagens, entre elas as de serem caros e pouco ergonômicos, exigindo a aquisição de tecnologias ainda pouco

acessíveis e muitas vezes limitadas aos centros de pesquisa de universidades e empresas, limitando a sua utilização pelo público em geral.

Sendo assim, o presente projeto de formatura se insere como uma alternativa de baixo custo e baixa dificuldade de implementação frente às aplicações de estereoscopia para RAE existentes hoje em dia.

1.2 OBJETIVOS DESTE TRABALHO

1.2.1 IMPLEMENTAÇÃO DE PROJEÇÕES ESTEREOSCÓPICAS EM AMBIENTE COM RAE UTILIZANDO SHADERS NO JAVA 3D

O objetivo primário deste projeto de formatura é aprimorar uma implementação existente de projeções estereoscópicas em ambiente com RAE, baseado no trabalho desenvolvido por Haseyama (2007). O trabalho realizado por Haseyama implementa uma técnica de projeção estereoscópica utilizando anaglifos, mas obteve um baixo desempenho em taxa de quadros. O baixo desempenho observado limita a aplicabilidade deste sistema, o que motivou a realização deste projeto. A implementação de Haseyama foi desenvolvida utilizando o Java 3D, porém sem o suporte a shaders programáveis oferecido por esta API, o que deixa espaço para uma melhora significativa na performance do programa em taxa de quadros do sistema.

Ao realizar a implementação desta técnica de projeção estereoscópica utilizando o suporte a shaders programáveis do Java 3D, espera-se obter um grande ganho na taxa de quadros atingida pelo programa atual. A utilização de shaders fará com que a maior parte do esforço de processamento exigido pelo programa seja efetuada no processador gráfico da placa de vídeo, justificando o alto ganho de desempenho esperado.

O elemento principal de que este projeto se utiliza para criar o efeito estereoscópico é a técnica de estereoscopia com anaglifos, que tem uma exigência de infra-estrutura menor do que outras técnicas de visão estéreo; a saber, um projetor do tipo *datashow* ou monitor e um par de óculos anaglíficos, que são óculos simples com filtros coloridos em cada uma das lentes.

1.3 ESTRUTURA DESTE TRABALHO

Este documento está dividido nas seguintes seções:

- **Capítulo 1 – Introdução:** Estabelecimento do contexto no qual o projeto se insere, exposição da motivação e justificativa para a sua realização, e definição de seus objetivos.
- **Capítulo 2 – Aspectos Conceituais:** Definição dos conceitos teóricos abordados pelo projeto, no nível de detalhe e aprofundamento necessário para a plena compreensão do desenvolvimento e dos resultados obtidos no projeto.
- **Capítulo 3 – Metodologia:** Descrição da metodologia de pesquisa e desenvolvimento utilizada na busca da solução proposta, inclusive das etapas do projeto desde a fase de pesquisa e levantamento de informações.
- **Capítulo 4 – Especificação do Projeto:** Plano conceitual da solução, isto é, o projeto de software propriamente dito. Nesta seção estão presentes documentos como diagramas de bloco e classe, definição de requisitos de hardware e software, especificação de funcionalidades, e quaisquer documentos adicionais julgados necessários para a adequada compreensão geral do sistema implementado.
- **Capítulo 5 – Projeto e Implementação:** Exposição detalhada das características de implementação do sistema. Contém explicações sobre o funcionamento específico das principais classes implementadas.
- **Capítulo 6 – Resultados Obtidos e Discussão:** Exposição e avaliação dos resultados alcançados pela implementação do sistema, indicando suas principais conquistas e apontando pontos passíveis de aprimoramento.
- **Capítulo 7 – Considerações Finais e Trabalhos Futuros:** Conclusão final sobre o desenvolvimento e os resultados finais do sistema e sugestões de direcionamento de eventuais trabalhos futuros, baseado nos pontos fracos levantados no capítulo anterior.

2. ASPECTOS CONCEITUAIS

Neste capítulo serão abordados os principais conceitos teóricos pertinentes a este projeto. Cada tópico é desenvolvido em um nível de detalhe suficiente para a compreensão do funcionamento geral do objeto deste projeto, bem como para o embasamento das escolhas feitas na realização do mesmo.

2.1 COMPUTAÇÃO GRÁFICA: HISTÓRIA E EVOLUÇÃO

A Computação Gráfica pode ser definida como a disciplina da Ciência da Computação cujo propósito é pesquisar e desenvolver técnicas computacionais de transformação de dados em imagens, e vice-versa. Hoje em dia a importância da computação gráfica se estendeu para boa parte dos setores da economia moderna. A multibilionária indústria do entretenimento, os setores de pesquisa científica de ponta, a indústria aeroespacial, o setor automobilístico, são apenas alguns exemplos de setores da economia que se beneficiam e dependem expressivamente do desenvolvimento da tecnologia da computação gráfica. (Sevo 2005)

Em certo sentido, a revolução do computador pessoal só foi possível nas proporções observadas hoje devido justamente ao desenvolvimento da computação gráfica. Esta evolução teve início com o projeto *Whirlwind*, um computador construído pelo MIT no final da década de 50 que foi considerado o primeiro computador digital capaz de operar em tempo real. Até então os computadores existentes funcionavam todos no modo *batch*, isto é, o processamento dos dados ocorria sempre de forma seqüencial, com a entrada dos dados, o processamento e a saída dos resultados ocorrendo em fases distintas e sem possibilidade de interferência no processamento após a entrada dos dados. O *Whirlwind* possuía uma interface de saída através de um tubo de raios catódicos (CRT), na época uma invenção recente (Sevo 2005).

Deste ponto em diante, a computação gráfica evoluiu rapidamente, e no final da década de 60 os primeiros avanços no campo de modelagem 3D por computador foram obtidos. Um dos primeiros dispositivos de visualização tridimensional de que se tem notícia foi justamente um dispositivo estereoscópico, um *Head Mounted Display* (HMD) desenvolvido por Ivan Sutherland (Sutherland 1969), o mesmo pesquisador do MIT que desenvolvera o *sketchpad* alguns anos antes. Enquanto o

sketchpad permitia ao seu usuário desenhar formas simples diretamente na tela do computador utilizando para isso uma caneta óptica, o HMD desenvolvido por Sutherland exibia duas imagens em *wireframe* separadas, uma para cada olho, criando assim uma ilusão tridimensional estereoscópica (Sevo 2005).

Com o advento do microprocessador no início da década de 70, a computação gráfica deu mais alguns saltos adiante. Em 1974 Ed Catmuli obteve o título de PhD em ciência da computação por sua tese sobre mapeamento de texturas, Z-buffer e superfícies curvas. Hoje em dia estas técnicas são utilizadas de maneira tão trivial que nem sequer nos damos conta do imenso avanço que as mesmas representaram na época. (Sevo 2005)

Um avanço importante ocorrido na década de 80 foi a introdução da utilização de pequenos trechos de código denominados *shaders*, que geram texturas através de fórmulas matemáticas, em contraste com o mapeamento de texturas baseado em imagens 2D existente até então. Além disso, os *shaders* podem ser facilmente separados para execução em processadores separados da CPU de um sistema, uma característica que seria integralmente explorada alguns anos mais tarde com o advento das placas gráficas (Sevo 2005).

Os anos 90 foram a década das superproduções cinematográficas que exploravam intensivamente os efeitos especiais criados por computador. Filmes como *Jurassic Park* (1994), *Toy Story* (1995), *O Máscara* (1995), para citar apenas alguns, obtiveram grande sucesso e impressionaram por seus efeitos visuais gerados por computador, impressionantes à época. Este período também marcou a ascensão do mercado de entretenimento eletrônico, com o explosão do mercado de placas gráficas para PC e de vídeo games como o *Playstation 1* (Sevo 2005).

Enquanto as décadas de 60 a 80 foram testemunhas da criação de boa parte dos conceitos e técnicas computacionais utilizadas até hoje, as duas últimas décadas foram o palco da aplicação e extensão destas técnicas ao seu pleno potencial, graças ao aumento exponencial na potência e capacidade dos computadores. O grau de realismo que vem sendo atingido pelos novos desenvolvimentos na área torna cada vez mais difícil distinguir o que é uma imagem real de uma imagem gerada 100% por computador. (Sevo 2005)

2.2 REALIDADE AUMENTADA

Ao contrário da Computação Gráfica, a RA é um campo de pesquisa ainda recente e pouco desenvolvido. Para se ter uma idéia, o termo RA foi utilizado pela primeira vez apenas em meados de 1990, apesar de alguns dispositivos de visualização estereoscópica e interação com objetos virtuais já existirem na ocasião. Na publicação original onde o termo foi pela primeira vez definido (Wellner, Mackay and Gold 1993), o mesmo foi introduzido como sendo o oposto da Realidade Virtual. Enquanto a Realidade Virtual imerge o usuário em um mundo sintético, formado puramente de informação, a meta da RA é amplificar o mundo real com a capacidade de processar informações (Wellner, Mackay e Gold 1993).

Desde os anos 90, o termo "RA" tem aparecido com freqüência na literatura científica. Entretanto, o seu uso se deu em diferentes contextos e com propósitos diversos, fazendo com que várias definições para o termo fossem formuladas, sem que nenhuma fosse especialmente mais aceita ou mais adequada (Milgram, et al. 1994).

Por exemplo, uma definição bem abrangente de RA seria "o aumento do *feedback* natural do usuário utilizando sinais simulados" (Milgram, et al. 1994). Uma outra definição desnecessariamente restritiva seria a de "uma forma de *realidade virtual* onde o *head-mounted display* do usuário é transparente, permitindo uma visão clara do mundo real" (Milgram, et al. 1994).

Podemos considerar a RA como parte de um universo mais amplo, a Realidade Misturada. Tomemos um espectro de dois extremos, o ambiente puramente real e o ambiente puramente virtual. Neste contexto, a realidade misturada consiste de um ambiente localizado entre estes dois extremos, onde objetos reais são apresentados juntamente com objetos virtuais em um mesmo dispositivo de exibição ou display. Dentro desta classificação as aplicações de realidade misturada podem ser caracterizadas em duas classes principais: a Virtualidade Aumentada, onde os elementos virtuais predominam sobre o mundo real e a RA, onde os elementos reais são predominantes sobre os elementos virtuais (Kirner e Tori 2004) (Milgram, et al. 1994).

A Figura 2-1 mostra o espectro de Realidade Virtual (Virtual Reality Continuum).

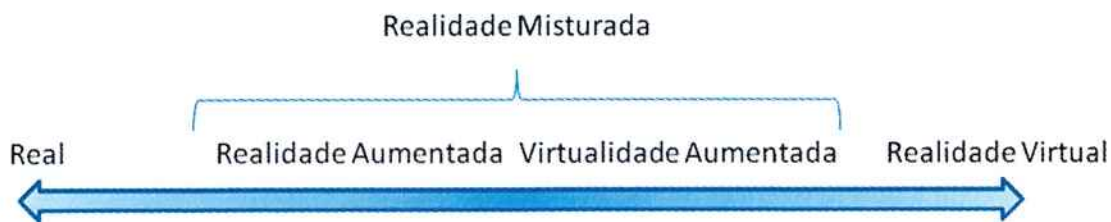

Figura 2-1: Espectro de Realidade Virtual. Adaptado de (Milgram, et al. 1994)

Entre os extremos deste espectro encontramos vários tipos de sistema; o sistema que vai nos interessar neste projeto é um sistema de RA baseado em uma definição adaptada de (R. T. Azuma 1997). Este sistema de RA possui as seguintes características:

- Combinação do ambiente real com elementos virtuais;
- Interação do usuário com os elementos virtuais em tempo real;
- Os elementos virtuais e o ambiente real devem estar alinhados espacialmente (isto é, possuir um sistema de coordenadas espaciais em comum) em três dimensões.

Esta definição retém os componentes essenciais da RA, sendo abrangente o suficiente para englobar uma variedade diversa de tecnologias e restritivo o suficiente para eliminar certas aplicações que não seriam de interesse do campo de RA. Filmes como "Jurassic Park", por exemplo, incluem em suas cenas objetos virtuais realistas misturados a um ambiente real em 3D, porém não constituem uma mídia interativa. A sobreposição bidimensional de objetos virtuais pode ser feita sobre vídeo em tempo real e com interatividade através de *Chroma-keying*, porém os objetos virtuais estão alinhados com o mundo real em apenas duas dimensões, não constituindo uma aplicação de RA. Entretanto, esta definição inclui interfaces baseadas em monitores, sistemas monoculares, HMDs, e várias outras técnicas de mistura de cenas (R. T. Azuma 1997).

Pelos motivos expostos acima, utilizaremos esta definição para o conceito de RA no restante deste projeto. A Figura 2-2 ilustra a combinação do ambiente real com elementos virtuais de acordo com a definição apresentada

acima.

Figura 2-2: RA com vaso e carro virtuais sobre a mesa (Tori, Kirner e Siscoutto 2006)

Os sistemas de RA apresentam requisitos mais restritos em comparação aos requisitos impostos pelas tecnologias de Realidade Virtual ou Virtualidade Aumentada desenvolvidas até hoje, como o posicionamento preciso dos elementos virtuais sobre a cena real (registro), boa percepção de profundidade, procedimentos simples de configuração inicial, baixa restrição da liberdade de movimentos do usuário, e baixa latência no cálculo e exibição de imagens. (State, et al. 1997)

VISÃO DIRETA X VISÃO INDIRETA

Segundo (Tori, Kirner e Siscoutto 2006), os ambientes com RA podem ser classificados em dois tipos principais, de acordo com a forma de visualização da cena misturada. Nos casos em que o mundo misturado é visualizado diretamente através dos olhos do usuário, a RA é classificada como imersiva ou de visão direta. Nos casos em que a mistura das cenas real e virtual ocorre em um dispositivo como um monitor ou projetor estático, a RA é classificada como não imersiva ou de visão indireta. Daí concluímos que a principal diferença entre os sistemas de visão direta e os de visão indireta diz respeito à solidariedade da superfície de projeção com relação ao observador. Nos sistemas de visão direta, a superfície de projeção é solidária ao observador, enquanto nos sistemas de visão indireta a superfície de projeção é estática.

O PROBLEMA DO REGISTRO

Um objetivo básico de qualquer sistema de RA é o alinhamento preciso dos objetos virtuais sobre os objetos reais, utilizando um sistema de coordenadas

tridimensionais comum aos dois universos. Isto exige, entre outras coisas, sensores que sejam precisos em tempo real para calcular a posição e orientação do usuário e dos objetos de interesse. Num cenário ideal, um sistema de RA deve ser capaz de efetuar registro com precisão milimétrica em ambientes externos e sem preparação prévia do ambiente. (Hoff e Azuma 2000)

Existem duas abordagens básicas que podem ser combinadas ou utilizadas separadamente para solucionar o problema do registro em aplicações de RA: através de sensores ou por técnicas de visão computacional, sendo que esta última vem ganhando bastante força com o aumento na disponibilidade de poder computacional vista nas últimas décadas. A visão computacional consiste de técnicas de reconhecimento de imagens por computador, que permitem obter informações sobre a posição de objetos cuja aparência possa ser modelada computacionalmente; desta forma, o vídeo captado pelas câmeras do sistema de RA é analisado em busca dos padrões de imagem que possibilitam o cálculo da posição dos objetos da cena. (Hoff e Azuma 2000), (Azuma, Neely, et al. 2006), (Hoff, Nguyen e Lyon 1996), (Lepetit 2008).

2.2.1 Realidade Aumentada Espacial

Dentro do campo de RA, a RAE abrange as técnicas de RA que realizam a mistura dos elementos reais e virtuais no próprio espaço físico do usuário, sem a utilização de dispositivos de visualização especiais como os *Head Mounted Displays*. Os elementos virtuais podem ser projetados sobre uma superfície física (Raskar 1998).

A detecção do ponto de vista do usuário permite que as imagens possam ser atualizadas dinamicamente criando a ilusão de que os elementos virtuais estão alinhados ao mundo real mesmo com o observador em movimento. (Raskar 1998)

Um dos grandes problemas encontrados por aplicações de RAE é a sua grande dependência das características da superfície de projeção. Uma superfície que possui baixa reflexão, por exemplo, dificulta a visualização das imagens. Existe também a interferência das sombras do próprio observador, que podem atrapalhar a projeção, agravando-se ao permitir que haja vários usuários no mesmo ambiente (Raskar 1998).

2.2.2 EXEMPLOS DE APLICAÇÕES DE RA: TÉCNICAS DE INTERAÇÃO

APLICAÇÃO MÉDICA

(Río, et al. 2005) descreve um sistema de interação para RA em uma aplicação médica, que utiliza pequenas esferas de superfície reflexiva para rastrear objetos que servem como ferramentas de interação com a aplicação. Uma ferramenta do tipo apontador é construída utilizando duas esferas colineares que definem um vetor no espaço, e uma ferramenta do tipo plano de corte é construída utilizando três esferas coplanares que definem um plano no espaço. Utilizadas em conjunto estas duas ferramentas permitem a seleção de pontos em três dimensões em um volume projetado no ambiente de RA.

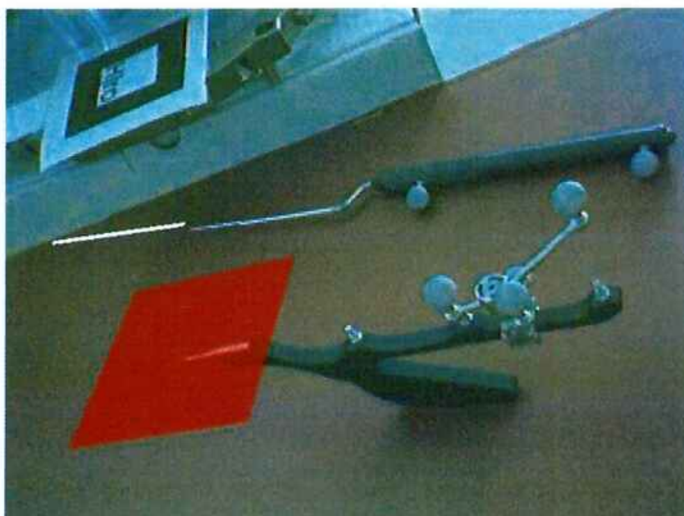

Figura 2-3: Ferramentas de Interação para Aplicação de RA: plano de corte (abaixo) e apontador (acima). (Río, et al. 2005)

PROJETO SUDIERSTUBE

O projeto Studierstube (Schmalstieg, et al. 2002) consiste de um sistema de RA colaborativo, com vários usuários participando do mesmo ambiente de RA simultaneamente. Um dos métodos de interação utilizados neste projeto consiste de um pequeno painel denominado Painel de Interação Pessoal (PIP). O PIP é uma interface controlada com as duas mãos consistindo de dois objetos leves, uma caneta e um painel, equipados com rastreadores. Os dois objetos são enxergados através de um HMD e aumentados com imagens geradas no computador, transformando-os em ferramentas de interação definidas pela aplicação. As

principais vantagens deste método são a intuitividade no uso da interface, o fato do usuário poder enxergar as próprias mãos, e o retorno tátil proporcionado pelo toque da caneta no painel. Esta forma de interação assimétrica com ambas as mãos explora o fato de que os humanos frequentemente utilizam a mão não-dominante para criar uma base de referência (segurar o painel) para as manipulações finas da mão dominante (controlar a caneta). Entretanto, o painel PIP oferece não só uma base de referência, mas também uma combinação natural de duas dimensões em 3D, como muitos dos objetos utilizados em nosso dia-a-dia (por exemplo controles remotos).

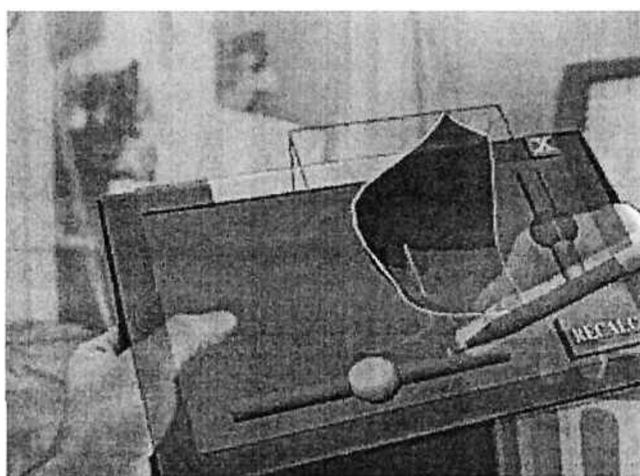

Figura 2-4: O Painel de Interação Pessoal permite a interação com widgets 2D e 3D utilizando as duas mãos. (Schmalstieg, et al. 2002)

PROJETO PINCHGLOVE

O sistema desenvolvido por (Veigl, et al. 2002) utiliza o ARToolkit para reconhecer marcadores presos a um par de luvas calçadas pelo usuário, além de um *touch screen* acoplado a uma das luvas, com sensores de pressão nos polegares atuando como botões. O mecanismo de interação obtido desta forma mostrou-se robusto e bastante flexível para aplicações de RA. A utilização da própria câmera do HMD para o reconhecimento dos marcadores colocados sobre as luvas possibilita o reconhecimento de outros marcadores presentes na cena aumentada com finalidades distintas.

Figura 2-5: Componentes do projeto Pinchglove, em sentido horário: a) Capacete HMD de combinação óptica com câmera para rastreamento óptico e um rastreador inercial; b) Luvas e touch pad utilizados em conjunto; c) Painel touch pad aumentado com interface da aplicação; d) Componentes da Luva Pinchglove. (Veigl, et al. 2002)

2.3 ESTEREOSCOPIA E VISÃO TRIDIMENSIONAL

Estereoscopia ou visão binocular é o resultado da interpretação feita pelo cérebro das imagens projetadas nos olhos. Devido à separação interocular, cada olho recebe uma projeção distinta de uma mesma cena observada, e isto nos proporciona a percepção de profundidade da cena. Ou seja, a partir das diferenças entre as duas imagens projetadas nas retinas, que são superfícies de projeção bidimensionais, o cérebro consegue extrair informações de profundidade e proporcionar uma percepção tridimensional dos objetos. (StereoGraphics Corporation 1997)

Esse fenômeno é apenas um dos métodos utilizados pelo ser humano para a percepção da profundidade. Outros aspectos, tais como nitidez das cores, oclusão de objetos, linhas de convergência, e etc., também são importantes na percepção de profundidade. O conjunto destes atributos da forma como os interpretamos nos dá a capacidade de visão tridimensional. (StereoGraphics Corporation 1997)

Em aplicações tridimensionais comuns, as imagens são geradas a partir de uma única câmera virtual e visualizadas em um monitor ou projetadas em um plano. As aplicações estereoscópicas se diferenciam por gerar duas imagens distintas correspondentes aos olhos esquerdo e direito do observador e apresentá-las separadamente a cada olho, simulando assim o efeito estereoscópico. (StereoGraphics Corporation 1997).

2.3.1 VISÃO TRIDIMENSIONAL

A seguir, explicamos os princípios que possibilitam a visão em três dimensões no ser humano.

CONVERGÊNCIA

Quando olhamos para um objeto, nossos dois olhos devem mover-se juntos, de maneira que ambos apontem diretamente para o objeto focalizado. Se o objeto estiver muito distante, os dois olhos apontam na mesma direção, isto é, focalizam em eixos aproximadamente paralelos (Mundo e Vestibular 2007).

Os raios de luz vindos do objeto distante são quase paralelos, e assim entrarão no olho e encontrarão a retina no ponto em que produzirão a imagem mais

nítida. Mas os olhos estão separados cerca de 6,4 centímetros (aproximadamente, em adultos) e assim, quando olhamos um objeto próximo, eles têm que se voltar para dentro para ver claramente e ainda permitir que a luz encontre a mesma parte sensível da retina (Mundo e Vestibular 2007)

Este processo de virar os olhos para ver um objeto próximo é chamado de convergência. As crianças podem "cruzar" seus olhos para enxergar objetos tão próximos quanto 7,5 cm, mas para adultos, a menor distância para uma visão clara é de 14 cm em média (Mundo e Vestibular 2007).

Sendo assim, cada olho vê uma figura ligeiramente diferente. Quanto mais perto o objeto maior serão as diferenças vistas por cada olho (Mundo e Vestibular 2007).

Para exemplificar o conceito da convergência, incluímos as figuras a seguir. A figura abaixo, extraída de (StereoGraphics Corporation 1997), exemplifica a percepção obtida por um observador com olhar fixo (foco) num objeto próximo (polegar), tendo ao fundo um objeto distante (bandeira).

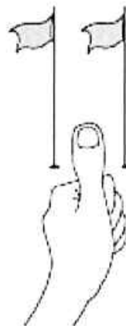

Figura 2-6: Imagem vista pelo observador com olhos fixos no objeto próximo

Já a próxima figura exemplifica a percepção obtida por um observador com foco num objeto distante (bandeira), tendo entre os olhos e o objeto focalizado um objeto mais próximo (polegar).

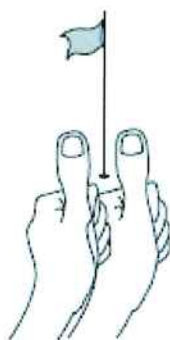

Figura 2-7: Imagem vista pelo observador com olhos fixos no objeto distante (StereoGraphics Corporation 1997)

PERSPECTIVA

O tamanho percebido de um objeto depende da distância do observador ao objeto; quanto mais distante um objeto, menor ele parecerá ao observador. Desta forma, dois objetos de mesmo tamanho podem aparentar tamanhos diversos de acordo com a distância em que se encontram em relação ao observador. Este efeito é denominado efeito de perspectiva, e é explorado extensivamente na criação de imagens com aparência tridimensional. Na figura a seguir, um ponto de fuga foi criado a partir das linhas de projeção existentes, criando o efeito da perspectiva. Os dois paralelepípedos parecem ter tamanhos diferentes, apesar de serem idênticos (Raposo, et al. 2004).

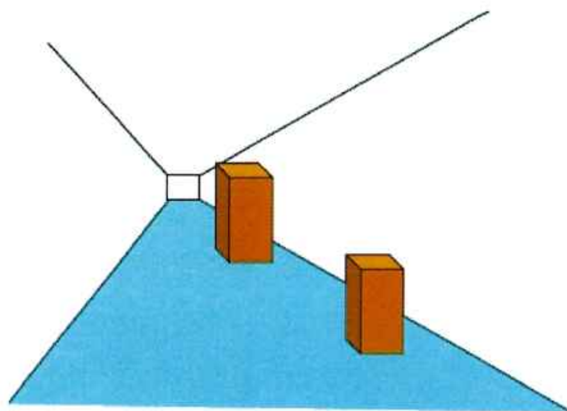

Figura 2-8: Paralelepípedos em perspectiva (Raposo, et al. 2004)

ILUMINAÇÃO

O grau de realismo de modelos tridimensionais criados por computador pode ser aumentado com técnicas de iluminação. A Figura 2-9 mostra um círculo e um hexágono preenchidos com cores de forma uniforme. Já na Figura 2-10 aplicou-se

um efeito de iluminação, criando uma percepção tridimensional maior dos objetos da cena (Raposo, et al. 2004).

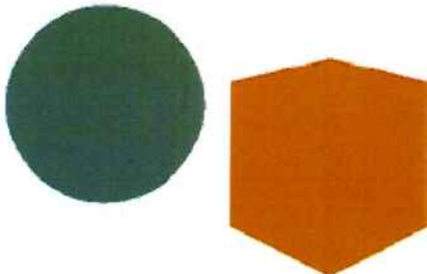

Figura 2-9: Círculo e Hexágono

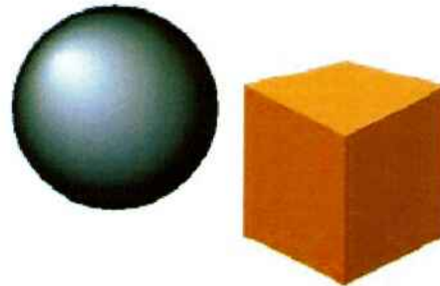

Figura 2-10: Esfera e Cubo

OCLUSÃO

Outra informação importante utilizada pelo sistema visual humano na percepção tridimensional é a oclusão, isto é, o encobrimento de partes de objetos que estejam por trás de outros objetos. (Raposo, et al. 2004).

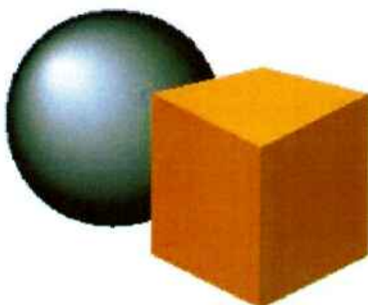

Figura 2-11: Esfera atrás do Cubo

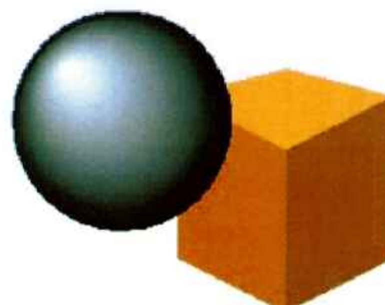

Figura 2-12: Esfera na frente do Cubo

SOMBRA

A projeção de sombras auxilia na percepção de que o objeto está ou não apoiado sobre um plano qualquer.

Se um objeto estiver a uma distância maior que zero da própria sombra, então podemos concluir que o objeto não está apoiado no plano, como ocorre com a esfera na figura abaixo (Raposo, et al. 2004).

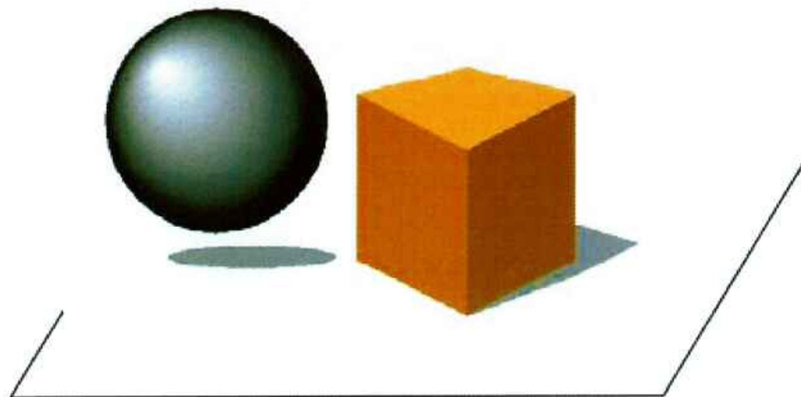

Figura 2-13: Efeitos da sombra

DISPARIDADE DA RETINA (ESTEREOSCOPIA)

A disparidade da retina é a distância horizontal entre pontos correspondentes das imagens de um objeto obtidas por cada um dos olhos. A disparidade acontece devido aos nossos olhos estarem separados a uma distância aproximada de 6,4 cm (em adultos) fazendo com que cada olho possua um ponto de vista diferente em relação ao objeto em questão (Raposo, et al. 2004).

A próxima figura representa de forma simplificada essa disparidade, onde a imagem mais a esquerda representa a visão do olho direito e a imagem mais a direita a visão do olho esquerdo. Se pudermos projetar cada uma das imagens separadamente nos olhos esquerdo e direito, teremos o efeito de estereoscopia ou visão binocular.

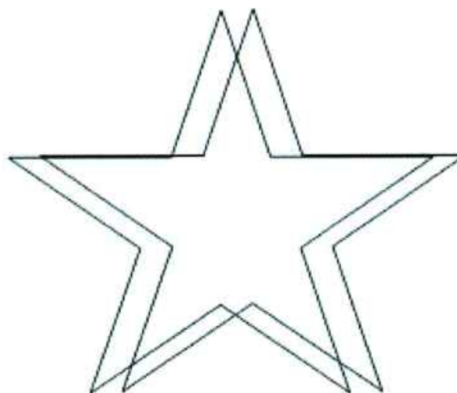

Figura 2-14: Disparidade da retina

PARALAXE

A paralaxe é a distância medida em relação a um anteparo de referência (que pode ser a tela do computador, por exemplo). As medidas de paralaxe podem ser classificadas em três tipos: zero, positiva e negativa (Raposo, et al. 2004).

- Paralaxe negativa: os eixos dos olhos se cruzam antes do anteparo (Figura 2-15).
- Paralaxe zero: os eixos dos olhos se cruzam no anteparo (Figura 2-16).
- Paralaxe positiva: os eixos dos olhos se cruzam após o anteparo (Figura 2-17).

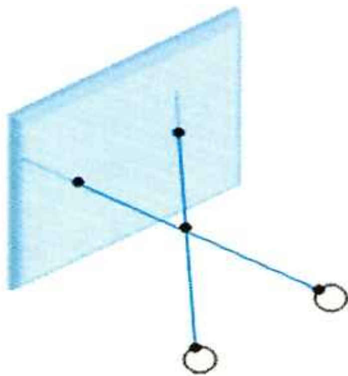

Figura 2-15: Paralaxe negativa

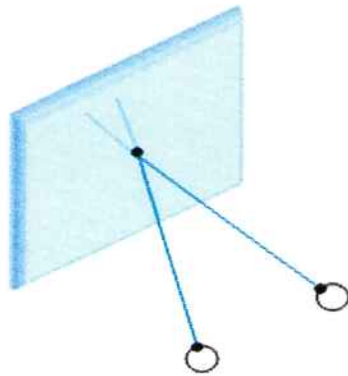

Figura 2-16: Paralaxe zero

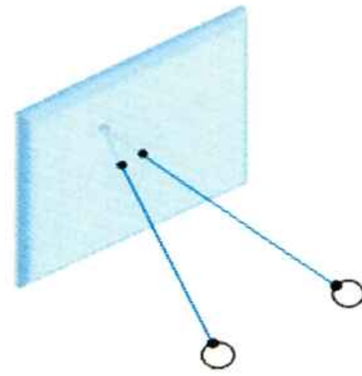

Figura 2-17: Paralaxe positiva

A paralaxe faz com que as imagens vistas por cada olho sejam diferentes provocando a disparidade da retina, que por sua vez provoca o fenômeno da estereoscopia (Raposo, et al. 2004).

Ela pode ser dada em termos de medida angular, relacionando-a com a disparidade e com a distância do observador a tela de exibição

Se as projeções de um ponto no olho esquerdo e direito estão separadas por P centímetros e o observador está a D centímetros da tela, então o ângulo de paralaxe α é dado por (ALBUQUERQUE 2006):

$$\alpha = \text{t}.$$

Em geral o valor do ângulo de paralaxe deve estar no intervalo [-1,5°, 1,5°], definindo valores de paralaxes mínimos e máximos. Valores fora deste intervalo tendem a causar desconforto na visualização (ALBUQUERQUE 2006).

A figura abaixo ilustra a fórmula indicada:

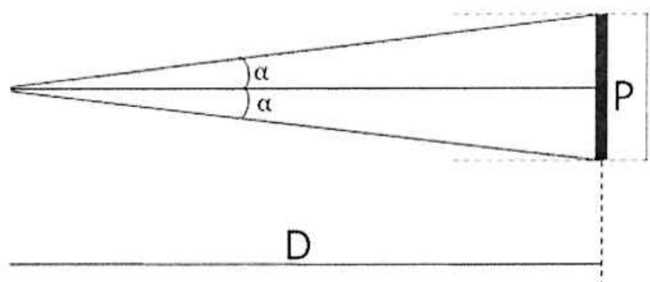

Figura 2-18: Valores de paralaxe

Se a paralaxe positiva tem um valor menor, mas próximo da distância entre os olhos o resultado é ruim, a menos que se queira posicionar o objeto no infinito. P não pode ser maior que a distância entre os olhos, pois neste caso os olhos teriam que divergir (Raposo, et al. 2004).

A estereoscopia não ocorre se apenas um olho enxergar o ponto. Para evitar que isso aconteça deve-se tomar cuidado para que as projeções de cada ponto de vista estejam contidas no retângulo que define o campo de visão no plano de projeção (Raposo, et al. 2004).

2.4 TÉCNICAS DE ESTEREOSCOPIA

A seguir, explicamos algumas das técnicas mais comuns para a produção artificial do efeito de estereoscopia. Existem duas classificações de equipamentos para a estereoscopia – estéreo passivo e estéreo ativo.

- Estéreo Passivo: As duas imagens são exibidas juntas e os óculos funcionam como filtros, separando cada imagem sobre o olho respectivo. Exemplo: anaglifo, polarização de luz.
- Estéreo Ativo – Nesse tipo de sistema os óculos são capazes de exibir em cada olho a sua imagem correspondente com a ajuda do sistema de exibição do estéreo. Exemplo: óculos obturadores, HMDs.

Independente do modo utilizado (ativo ou passivo), a separação das imagens para cada olho requer um dispositivo de visualização que variam desde simples

óculos com filtros até óculos com obturadores eletrônicos, utilizados nas técnicas mais sofisticadas (Raposo, et al. 2004).

PAR ESTEREOSCÓPICO

Um par estereoscópico é um conjunto de duas imagens bidimensionais tomadas de uma mesma cena, com uma pequena separação horizontal entre os pontos de captura da imagem, simulando o espaçamento entre os olhos humanos. O princípio básico de todos os métodos de produção artificial de estereoscopia consiste em obter ou criar as duas imagens que compõem um par estereoscópico e apresentar separadamente cada imagem do par ao seu olho correspondente (direito ou esquerdo).

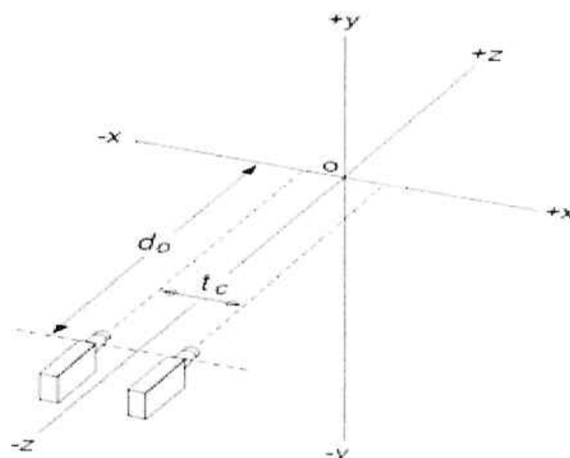

Figura 2-19: Obtenção de um par estereoscópico – separação horizontal entre as câmeras

2.4.1 ÓCULOS POLARIZADOS

O fato dos raios de luz se comportarem como ondas transversais, isto é, que oscilam em planos perpendiculares à direção de propagação, possibilita a criação der uma técnica de estereoscopia bastante interessante. Através da polarização da luz, podemos criar feixes de luz que se propagam vibrando apenas em um determinado plano, e.g., vertical ou horizontal. Se construirmos as imagens de um par estéreo, sendo uma imagem polarizada verticalmente e a outra polarizada horizontalmente, a utilização de óculos com lentes polarizadas fará com que apenas uma das imagens atinja cada olho, obtendo o efeito estereoscópico.

Uma forma de implementação deste princípio utiliza uma tela coberta com um painel polarizador de luz, onde as duas imagens são apresentadas simultaneamente

na tela, sendo a primeira imagem formada apenas pelas linhas pares da tela e a segunda imagem construída nas linhas ímpares. O painel polarizador faz com que as linhas pares sejam polarizadas em uma direção ortogonal à polarização que é submetida às linhas ímpares e a utilização de óculos polarizados permite que cada olho apenas receba a imagem que lhe corresponde (TOMMASELLI 2006).

Nesta técnica, as lentes dos óculos são feitas com elementos passivos de fácil construção apresentando assim, custo, tamanho e peso reduzidos. A maior dificuldade desta técnica está na utilização da placa polarizadora montada sobre a tela, devido ao seu custo elevado (TOMMASELLI 2006).

A figura abaixo, extraída de (Zalman Cool Innovations 2007), ilustra de forma simplificada o funcionamento da técnica.

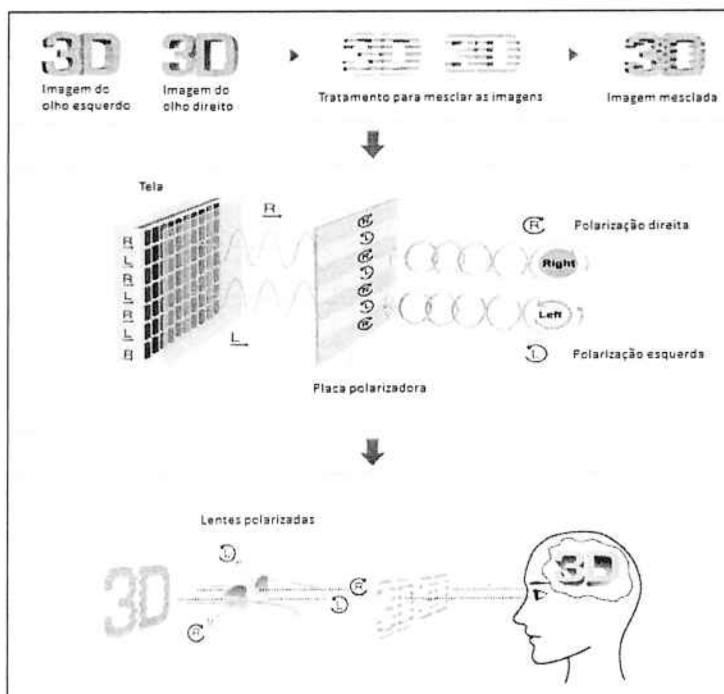

Figura 2-20: Esquema simplificado da técnica de lentes polarizadas

2.4.2 ÓCULOS OBTURADORES

O olho humano consegue discernir dois eventos luminosos consecutivos que ocorram a intervalos de no mínimo 1/10 s. É por este motivo que as imagens no cinema são projetadas com duração de 1/24 s e percebidas como contínuas. Isso ocorre também na televisão, onde os quadros têm a duração de aproximadamente 1/30 s (ANDRADE 2007).

A utilização de óculos obturadores consiste na captura das imagens referentes a cada olho, gerando um vídeo denominado Vídeo Estereoscópico Campo-Sequencial. Este vídeo é exibido por um projetor ou monitor que utiliza um único sinal com o dobro da freqüência padrão, alternando a cada 1/60 s entre a exibição dos campos esquerdo e direito da imagem. Ou seja, a cada 1/30 s projetam-se duas imagens com a duração de 1/60s, uma para cada olho (ANDRADE 2007).

Para a visualização do efeito são utilizados óculos obturadores de cristal líquido (*LCD shutter glasses*), cujas lentes possuem a capacidade de ficar instantaneamente transparentes ou opacas de acordo com um sinal eletrônico, permitindo ou impedindo a passagem de luz em cada uma das lentes dos óculos. As lentes dos óculos abrem e fecham alternadamente a cada 1/60 s, sincronizando-se com o vídeo sendo exibido, de tal forma que a abertura do lado esquerdo dos óculos corresponda à projeção do campo referente ao olho esquerdo, e o mesmo para o lado direito (ANDRADE 2007).

Figura 2-21: Óculos obturadores

A principal desvantagem deste sistema é o custo elevado do projetor e dos óculos. O projetor utilizado deve operar em frequências mais altas que os projetores comuns. A qualidade das imagens é a muito superior à de outras técnicas de visão estéreo, constituindo uma grande vantagem.

2.4.3 ANAGLIFOS

A criação do efeito de estereoscopia utilizando anaglifos é feita pela separação das imagens por cores. Cada imagem do par estéreo é composta por uma ou mais cores primárias de forma que a imagem esquerda não possua uma das

cores da imagem direita e vice-versa. Com este princípio conseguimos separar as imagens esquerda e direita utilizando óculos com filtros coloridos, específicos para essa finalidade (ALBUQUERQUE 2006).

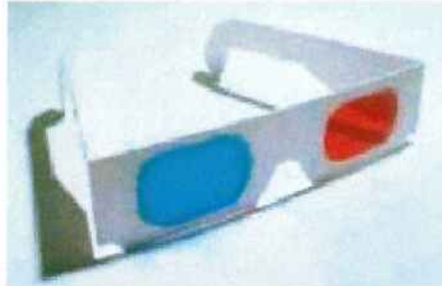

Figura 2-22: Óculos utilizados para visão estéreo com anaglifo (TORI et al., 2006).

Cada um dos olhos enxerga a imagem através de um filtro diferente, recebendo apenas uma das imagens do par estereoscópico. O filtro vermelho deixa atingir o olho apenas a imagem vermelha do anaglifo e o filtro azul/verde transmite ao olho apenas a imagem azul/verde. A Figura 2-23 ilustra essa separação de cores. A Figura 2-24 mostra um exemplo de anaglifo.

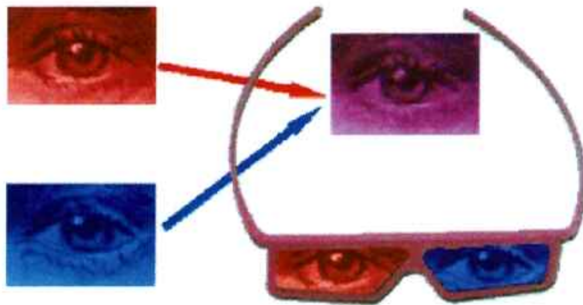

Figura 2-23: Separação de cores através dos óculos anaglíficos (Raposo, et al. 2004)

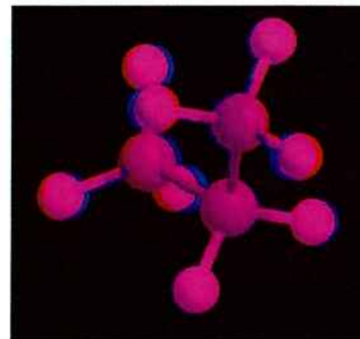

Figura 2-24: Exemplo de anaglifo (Raposo, et al. 2004)

Exibir um anaglifo requer apenas um projetor ou monitor e um par de óculos anaglíficos. O anaglifo pode ser impresso e é uma solução de baixo custo. A principal desvantagem é a perda de qualidade na imagem devido à perda de cores, fator este que pode ser atenuado conforme será exposto adiante.

Dada a proposta deste projeto, de ser uma implementação de estereoscopia acessível e de baixo custo, a técnica de anaglifos apresenta diversas vantagens interessantes que justificam a sua adoção, por exemplo:

- Baixo custo de reprodução e visualização, podendo ser exibida por monitores ou projetores comuns e visualizados através de óculos com filtros coloridos simples e de fácil confecção;
- O efeito estéreo gerado pelo anaglifo independe da mídia de visualização, podendo ser impresso, projetado, estático, animado, etc;
- A perda de qualidade causada pela coloração pode ser atenuada utilizando-se modos de anaglifo colorido, que conseguem uma reprodução parcial das cores da imagem original.

Devido a estes fatos, a estereoscopia utilizando anaglifos foi o método selecionado para este projeto. A seguir fornecemos detalhes sobre as diferentes fórmulas para produção de anaglifos, e suas características.

MÉTODOS DE PRODUÇÃO DE ANAGLIFOS

A produção de um anaglifo no computador consiste na obtenção das duas imagens do par estereoscópico e na coloração adequada de cada uma das imagens pixel a pixel, com a posterior soma dos pixels das duas imagens. Existem alguns métodos diferentes para a obtenção de uma imagem em estéreo com anaglifo, e cada um apresenta certas características, vantagens e desvantagens.

A seguir damos uma descrição dos modos de anaglifo implementados neste projeto, explicando os detalhes de cada modo. São dadas as fórmulas para o cálculo dos valores de cor r_a, g_a, b_a do anaglifo, em RGB, a partir das imagens esquerda (r_1, g_1, b_1) e direita (r_2, g_2, b_2). As fórmulas devem ser aplicadas para cada pixel. Salvo indicação em contrário, todas as referências, fórmulas e figuras da presente seção foram obtidas em (3dtv.at 2005).

ANAGLIFO PURO

Consiste na remoção de todas as cores da imagem original, aplicando aos componentes vermelho e azul da imagem final quantidades proporcionais de cada componente RGB das imagens esquerda e direita, respectivamente. A fórmula para o cálculo das imagens é a seguinte:

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0{,}299 & 0{,}587 & 0{,}114 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

Este método possui as seguintes características:

- Escurecimento da imagem;
- Perda das cores;
- Pouco vazamento de cor nos filtros.

O vazamento de cor ocorre quando a imagem de um olho não pode ser filtrada completamente pelo seu filtro correspondente, resultando em porções da imagem esquerda sendo enxergadas pelo olho direito, e vice-versa. Está associado à qualidade dos filtros utilizados e também à proporção dos canais verde e azul presentes na imagem esquerda (correspondente ao filtro vermelho), e vice-versa.

Figura 2-25: Anaglifo puro

ANAGLIFO EM ESCALA DE CINZA

Este método acrescenta à imagem final uma componente no canal verde, composta proporcionalmente pelas cores da imagem original direita, resultando em uma imagem final em escala de cinza. A imagem é calculada pela seguinte fórmula:

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}299 & 0{,}587 & 0{,}114 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

Este método apresenta as seguintes características:

- Perda das cores, porém com a imagem em escala de cinza;
- Mais vazamento do que em imagens com anaglifo puro.

Figura 2-26: Anaglifo em escala de cinza

ANAGLIFO EM CORES

Este método consiste na utilização integral das componentes vermelha, azul e verde da imagem original, sendo a componente vermelha tomada da imagem esquerda e as componentes azul e verde tomadas da imagem direita:

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

Utilizando a fórmula acima, obtemos um anaglifo com:

- Reprodução parcial de cores;
- Aparecimento de rivalidade binocular, ou rivalidade entre retinas.

A rivalidade binocular ocorre quando o cérebro se recusa a interpretar separadamente as informações recebidas em cada olho, fazendo com que as cores dos objetos pareçam saltar ou piscar durante a visualização da imagem. Isto ocorre devido à supressão alternada de cada imagem, ou seja, o cérebro considera apenas uma das imagens recebidas de cada vez. Eventualmente o olho dominante irá determinar qual imagem será interpretada, porém este fenômeno pode provocar incômodo e fadiga durante a visualização. (Sousa 2006)

Nos anaglifos, a rivalidade retínica ocorre em dois níveis: devido às diferenças nas cores captadas por cada olho, e devido às diferenças de brilho e contraste existentes entre as imagens esquerda e direita após a modificação das cores.

Figura 2-27: Anaglifo em cores

ANAGLIFO SEMI-COLORIDO

Este método é derivado do método anterior, reproduzindo em parte as componentes azul, verde e vermelha da imagem esquerda, e mantendo intactos os canais azul e verde da imagem direita.

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

Os anaglifos semi-coloridos apresentam as seguintes propriedades em relação aos anaglifos em cores:

- Reprodução parcial de cores, porém menor do que nos anaglifos em cores;
- Menos rivalidade retínica do que nos anaglifos em cores.

Figura 2-28: Anaglifo semi-colorido

ANAGLIFO OTIMIZADO

Este método aplica à imagem final parte dos canais verde e azul da imagem esquerda, mantendo inalterados os componentes azul e verde da imagem direita.

$$\begin{pmatrix} r_a \\ g_a \\ b_a \end{pmatrix} = \begin{pmatrix} 0 & 0,7 & 0,3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} r_1 \\ g_1 \\ b_1 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} r_2 \\ g_2 \\ b_2 \end{pmatrix}$$

Este método permite obter:

- Reprodução parcial de cores, porém perdendo os tons de vermelho do original;
- Rivalidade retínica bastante reduzida.

Figura 2-29: Anaglifo otimizado

2.5 O PIPELINE GRÁFICO DO OPENGL

O OpenGL (*Open Graphics Library*) é uma interface de software para manipulação do hardware gráfico de um sistema. Ele expõe funcionalidades avançadas de renderização do hardware gráfico mantendo um modelo de programação simples. (Segal e Akeley 2006)

Sob a ótica do programador, o OpenGL é um conjunto de comandos que permite a especificação de objetos em duas ou três dimensões, juntamente com comandos que controlam a renderização destes objetos no framebuffer. (Segal e Akeley 2006)

Sob a ótica do implementador (fabricante do hardware gráfico), o OpenGL é um conjunto de comandos que afeta a operação do hardware gráfico. Se o hardware

consistir de apenas um framebuffer manipulável, o OpenGL deve ser implementado em sua maior parte na CPU do sistema. Tipicamente, o hardware gráfico pode incluir componentes de aceleração gráfica em graus variados, desde um subsistema de rasterização capaz de renderizar linhas e polígonos bidimensionais até sofisticados processadores de ponto flutuante capazes de transformar e computar dados geométricos. A função do implementador do OpenGL é fornecer a interface de software especificada, dividindo o esforço necessário para cada comando do OpenGL entre a CPU e o hardware gráfico. (Segal e Akeley 2006)

Sob a ótica dos seus criadores, o OpenGL é uma máquina de estados que controla um conjunto de operações de renderização. O modelo de máquina de estados deve oferecer uma especificação satisfatória para os programadores e implementadores, mas não é necessariamente um modelo de implementação. Uma implementação deve ser capaz de fornecer os resultados da maneira como especificados em (Segal e Akeley 2006), porém a forma de execução das computações não é de forma alguma ditada por esta especificação.

A funcionalidade principal oferecida pelo OpenGL consiste na transformação de primitivas geométricas (ponto, linha, polígono, bitmap, etc) em pixels, renderizando-os no framebuffer do hardware gráfico que é então exibido em tela. A figura abaixo mostra o pipeline de processamento do OpenGL: (Fernandes 2006)

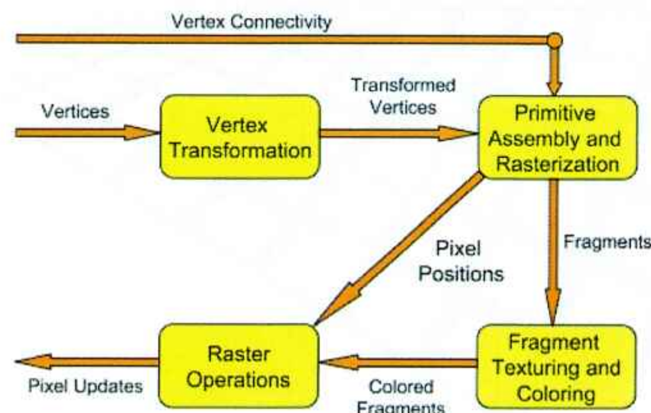

Figura 2-30: O Pipeline gráfico do OpenGL - funcionalidade fixa

O pipeline indicado na figura mostra a funcionalidade fixa do OpenGL, que é a funcionalidade padrão oferecida pela API. Mais adiante veremos que esta funcionalidade pode ser modificada com o uso de shaders programáveis, que substituem as operações fixas de certos blocos da estrutura do pipeline.

A sequência de operações executada em cada bloco do pipeline é a seguinte: (Fernandes 2006)

- Transformação de vértices: recebe as coordenadas e propriedades de cada vértice, bem como o estado do OpenGL, e retorna o vértice transformado pelas matrizes de visualização (ModelView) e projeção (Projection); aplica iluminação, e gera e/ou transforma coordenadas de textura.
- Montagem de primitivas: recebe os vértices transformados e as informações de conectividade dos vértices. Retorna as primitivas gráficas (polígonos), efetuando os cortes da cena de acordo com o volume de visualização (*view frustum*), e descartando as primitivas invisíveis (*back face* e *Z culling*).
- Rasterização: Recebe as primitivas construídas no passo anterior e determina o conjunto de pixels cobertos por cada primitiva, bem como os atributos de cada pixel, por interpolação dos atributos entre os vértices adjacentes. Retorna um conjunto de fragmentos.
- Texturização e coloração: Esta fase recebe os valores interpolados para cada fragmento e realiza a aplicação de texturas e coloração, calculando a cor final do fragmento.
- Rasterização: A fase final do pipeline realiza operações como mistura (blending) da cor do fragmento com a cor existente em um buffer de cor, testes alpha, stencil, de profundidade, etc.

A figura abaixo fornece um resumo visual deste processo:

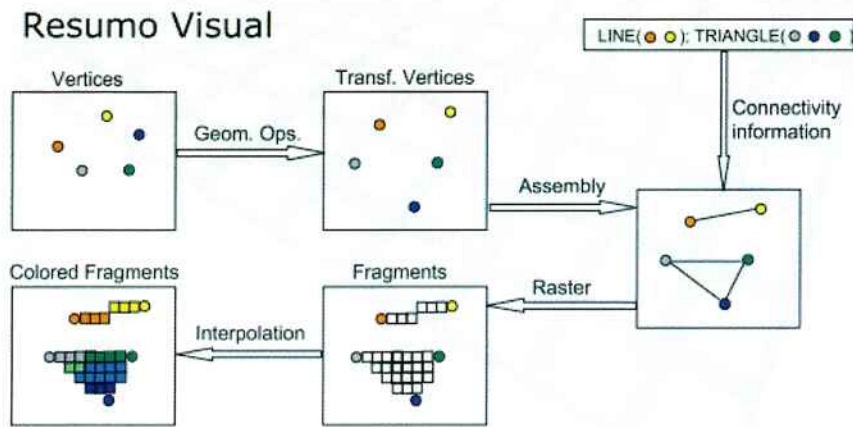

Figura 2-31: Resumo visual do processamento realizado pelo pipeline gráfico do OpenGL (Fernandes 2006)

A partir de sua versão 2.0, o OpenGL incorporou definitivamente o suporte a shaders programáveis. Um shader é um programa capaz de operar sobre os vértices e fragmentos do OpenGL. O uso de um shader programável substitui efetivamente certos blocos da funcionalidade fixa do pipeline gráfico. Os blocos cuja funcionalidade pode ser substituída são o processador de vértices e o processador de fragmentos, como ilustra a figura a seguir:

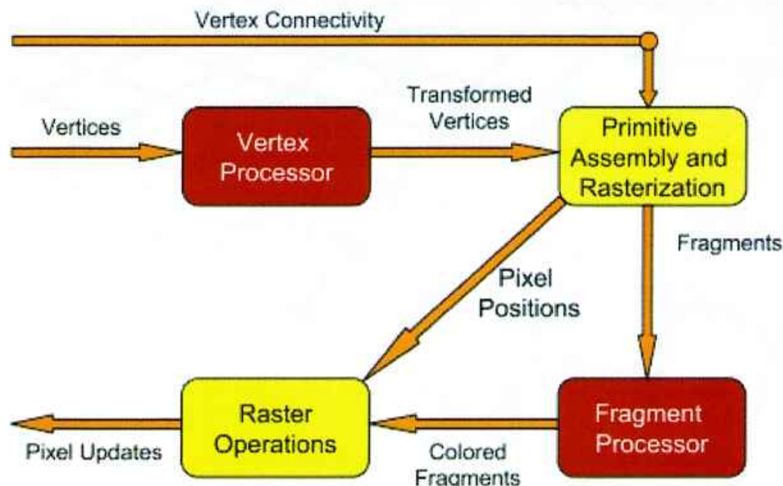

Figura 2-32: Pipeline do OpenGL com funcionalidade fixa substituída por shaders programáveis (Fernandes 2006)

O OpenGL 2.0 oferece ao programador a possibilidade de substituir as fases de processamento de vértices e processamento de fragmentos implementadas pela funcionalidade fixa. Entretanto, esta substituição remove completamente a funcionalidade fixa do OpenGL, e o programador deve fornecer suporte a todas as

operações da funcionalidade fixa que deseje implementar, e.g., iluminação de vértices ou texturização de fragmentos. (Fernandes 2006)

O processador de vértices é uma unidade programável que opera sobre os vértices de entrada e seus atributos. Um programa que execute neste processador é chamado de *vertex shader*. O processador de vértices opera sobre um vértice de cada vez. Ele não pode substituir operações gráficas que exijam o conhecimento de mais de um vértice por vez. Os vertex shaders que forem executados no processador de vértices devem computar a posição homogênea do vértice de entrada. (John Kessenich 2006)

O processador de fragmentos é uma unidade programável que opera sobre os valores dos fragmentos e seus dados associados. Um programa executado neste processador é chamado de *fragment shader*. Um fragment shader não pode alterar a posição de um fragmento. Não é permitido o acesso aos fragmentos vizinhos. Os valores computados por um fragment shader são utilizados para atualizar a memória do framebuffer ou a memória de textura, de acordo com o estado atual do OpenGL. (John Kessenich 2006)

Uma aplicação OpenGL pode substituir apenas uma das funcionalidades (*vertex* ou *fragment processor*), utilizando a funcionalidade fixa fase do pipeline não substituída. A figura abaixo mostra os passos necessários em uma aplicação OpenGL para o uso de shaders programáveis (Fernandes 2006). O Java3D automatiza este procedimento, encapsulando estas operações nas entidades de alto nível detalhadas na seção *API Java 3D*.

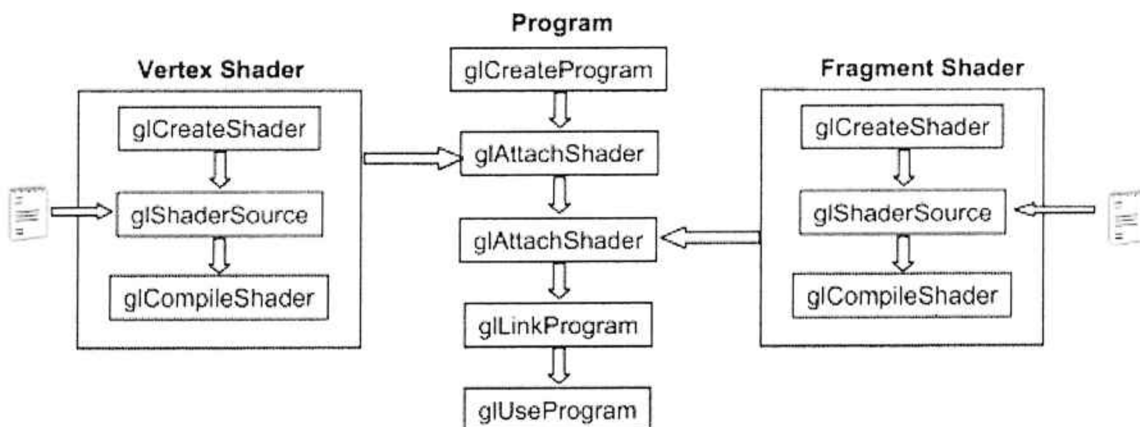

Figura 2-33: Processo de substituição da funcionalidade fixa em uma aplicação OpenGL

O GLSL é a linguagem oficial do OpenGL para a criação de shaders programáveis. Ela é baseada na linguagem ANSI C, com algumas funcionalidades do C++, e o acréscimo de suporte nativo a tipos de dados de vetores e matrizes.

2.6 A API JAVA 3D

A API Java 3D é uma hierarquia de classes Java que possibilita o desenvolvimento de sistemas gráficos tridimensionais utilizando o formato consagrado pela linguagem Java, cujo lema é "*write once, run anywhere*"; isto é, um programa precisa ser escrito apenas uma vez e pode ser executado em qualquer plataforma.

Diferente da biblioteca JOGL (Java OpenGL), que simplesmente encapsula as chamadas de função do OpenGL dentro do Java, o Java3D implementa uma funcionalidade completa de desenvolvimento gráfico tridimensional orientado a objetos, através de sua hierarquia de classes baseada em grafos de cena.

Esta hierarquia de classes possibilita manipular objetos tridimensionais, objetos de iluminação, som, planos de fundo e diversos outros elementos utilizados na criação de universos virtuais em três dimensões. O universo virtual é construído em um grafo de cena, uma estrutura de dados em forma de árvore que é composta dos objetos gráficos e do sub-grafo de visualização (Manssour 2003).

O Java 3D encapsula as bibliotecas OpenGL e Direct3D, implementando uma camada de abstração no topo destas bibliotecas que oferece entidades de alto nível para que o desenvolvedor não precise se preocupar com os detalhes de renderização da cena, que são gerenciados automaticamente pelas classes de visualização do Java 3D. A Figura 2-34 ilustra a organização em camadas do Java 3D.

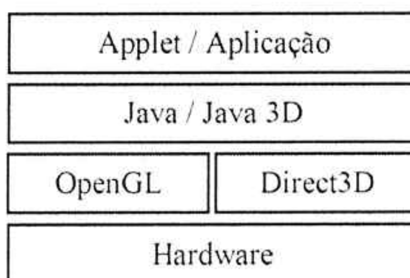

Figura 2-34: Organização em camadas de um sistema utilizando Java3D (Manssour 2003)

2.6.1 GRAFOS DE CENA

O grafo de cena é a estrutura principal de um aplicativo Java3D. Um grafo de cena contém dados organizados hierarquicamente em uma árvore. O grafo de cena é composto por nós ancestrais (pais), nós terminais (filhos) e objetos de dados. Os nós ancestrais, ou *group nodes*, organizam e controlam como o Java 3D interpreta os seus descendentes. Os nós filho podem ser *group nodes* ou *leaf nodes*, sendo que os *leaf nodes* são nós terminais e não têm filhos. Estes nós terminais determinam os aspectos semânticos principais de um grafo de cena – geometria, áudio, objetos de iluminação e comportamento, e assim por diante. Os nós terminais referenciam objetos de dados do tipo `NodeComponent`, que não são nós do grafo de cena, mas contém os dados utilizados pelos nós terminais, como a geometria a ser renderizada ou as amostras de sons executadas. (Sun Microsystems 2001)

Um grafo de cena é criado a partir de instâncias de *group nodes* e *leaf nodes*. A estrutura típica de um grafo de cena é mostrada na Figura 2-35. Neste diagrama podemos perceber como os sub-grafos de conteúdo e visualização são separados, simplificando a manipulação da plataforma de visualização e do conteúdo adicionado ao grafo de cena.

Os objetos correspondem aos nós, ou vértices, do grafo de cena e os relacionamentos entre estes objetos são representados pelas arestas do grafo. Os relacionamentos podem ser do tipo "referência", associando um objeto ao grafo, ou "herança", onde um nó do tipo 'Grupo' pode ter um ou mais filhos e apenas um pai, e um nó do tipo 'Folha' não pode ter filhos (Manssour 2003).

A representação padrão de grafos de cena utiliza círculos para os nós do tipo grupo, enquanto nós do tipo folha são representados por triângulos. Os demais objetos da cena são representados por retângulos (Manssour 2003).

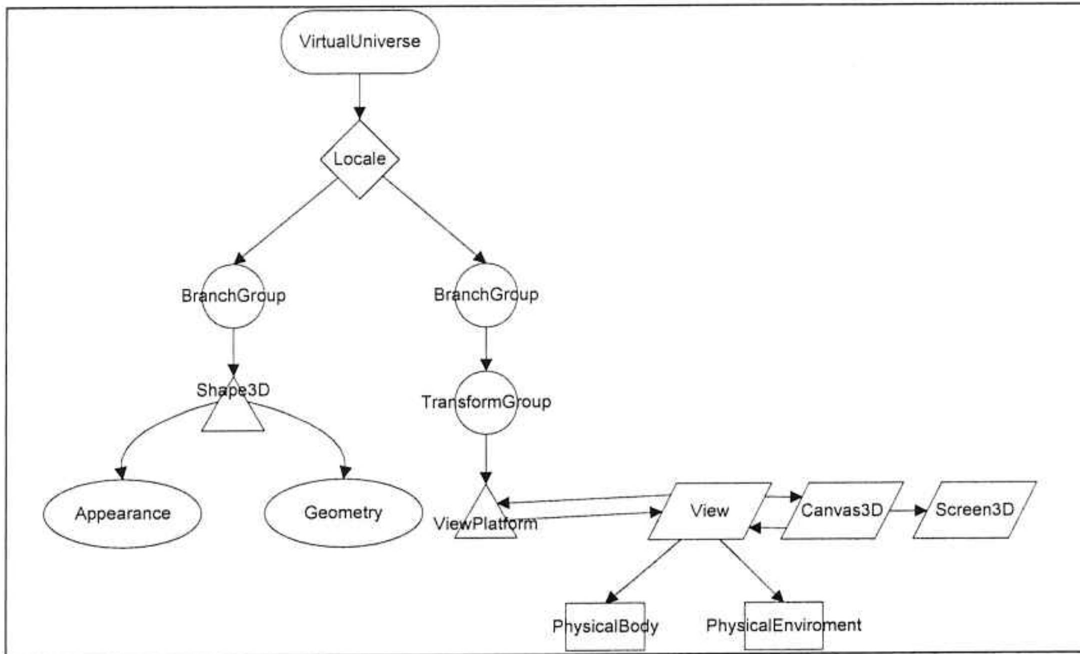

Figura 2-35: Grafo de cena do Java3D. (Manssour 2003)

Um grafo de cena deve conter apenas um nó `VirtualUniverse`, que define um universo virtual e possui pelo menos um objeto `Locale`, responsável pela especificação de um ponto de referência no universo virtual e raiz dos sub-grafos de um grafo de cena (Manssour 2003).

Os nós do tipo `BranchGroup` servem para agrupar os nós de uma cena relacionados através de alguma associação ou conjunto de características em comum. (Manssour 2003).

Os sub-grafos abaixo do nó `Locale` são dois: o sub-grafo de conteúdo, que descreve geometrias, aparências, comportamentos, sons e luzes, ou seja, o conteúdo do universo virtual; e o sub-grafo de visualização, que controla a visualização da cena através de parâmetros como a localização do ponto de vista, a matriz de projeção, etc. (Manssour 2003)

Os nós do tipo `TransformGroup` especificam a posição, orientação e escala dos objetos no universo virtual, relativo a um nó `Locale` (Manssour 2003).

Os nós do tipo `Behavior`, não representado na figura, contém o código necessário para manipular dinamicamente a matriz de transformação da geometria de um objeto. Utilizando este tipo de objeto podemos impor transformações dinâmicas aos atributos (posição, tamanho, cor, etc.) dos objetos. Estas

transformações podem ser condicionadas ao pressionamento de uma tecla, ao movimento do mouse, a uma interpolação no tempo, etc (Manssour 2003).

O nó `Shape3D` define as formas geométricas do universo virtual. Ele sempre faz referência a um ou mais objetos `Geometry`, que definem as formas geométricas descritas por este objeto, e a um objeto `Appearance`, que descreve a aparência desta geometria, com parâmetros como cor, textura, propriedades de reflexão da superfície, entre outras (Manssour 2003).

O nó `ViewPlatform` e seus objetos de dados associados descrevem o ponto de vista do observador dentro do universo virtual, e os parâmetros de renderização da cena na tela. (Manssour 2003)

2.6.2 Classes Principais do Java3D

Para a compreensão de alguns aspectos de implementação deste projeto, fornecemos abaixo alguns detalhes relevantes de certas classes do Java 3D utilizadas pela aplicação.

View

O objeto `View` contém todos os parâmetros utilizados na renderização de uma cena tridimensional a partir de um ponto de vista. Ele contém uma lista de objetos `Canvas3D` nos quais esta vista é renderizada. Este objeto não faz parte do grafo de cena, porém acopla-se a um nó `ViewPlatform` do grafo de cena.

Um objeto `View` pode ser ativado e desativado pela aplicação em tempo de execução, sendo que a renderização de um `Canvas3D` só é feita quando o seu objeto `View` associado está ativo. Este objeto também permite iniciar e interromper o agendamento de *Behaviors*, efetivamente interrompendo a atualização de animações e interações neste objeto. Estas duas capacidades são exploradas neste projeto.

SimpleUniverse

A classe `SimpleUniverse` é uma classe utilitária que automatiza a montagem do sub-grafo de visualização associado a um `VirtualUniverse`. A instanciação de um objeto `SimpleUniverse` estabelece a configuração de um

ambiente mínimo para executar um programa Java 3D, fornecendo as funcionalidades necessárias para a maioria das aplicações. Quando uma instância de `SimpleUniverse` é criada, ela gera automaticamente os objetos que compõem o sub-grafo de visualização, como `Locale`, `ViewPlatform` e `Viewer`.

A Figura 2-36 mostra o encapsulamento oferecido pela classe `SimpleUniverse`.

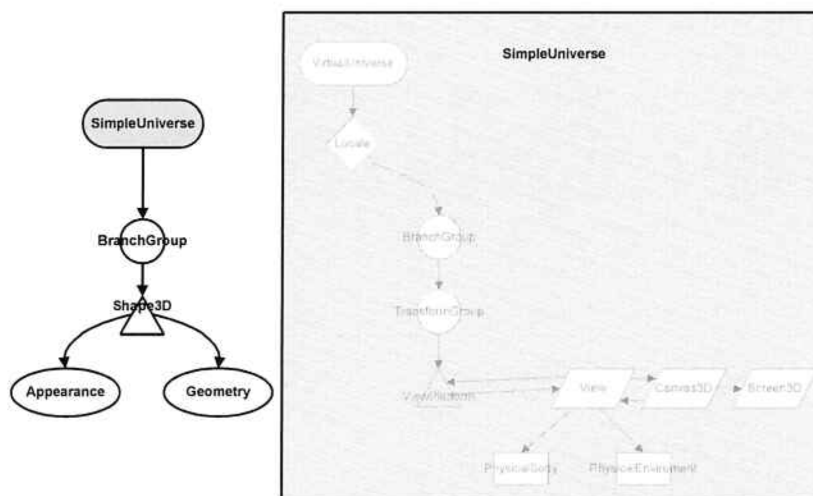

Figura 2-36: Encapsulamento oferecido pela classe `SimpleUniverse` (Haseyama 2007)

APPEARANCE

O objeto `Appearance` define os parâmetros de renderização que podem ser especificados como componentes de um nó `Shape3D`. Este parâmetros incluem: coloração, transparência, textura, entre outros. Uma subclasse importante de `Appearance` para este projeto é a classe `ShaderAppearance`, que permite a aplicação de um shader programável ao seu objeto `Shape3D` associado. A hierarquia de classes que permite o uso de shaders no Java 3D é explicada em detalhes mais adiante. (Sun Microsystems 2003)

CANVAS3D

`Canvas3D` é uma classe bastante complexa, pois é ela que implementa a renderização da cena em tela, possuindo diversos modos de operação e possibilidades de configuração. Por sua importância para este projeto, explicamos detalhadamente o funcionamento desta classe abaixo.

O `Canvas3D` é uma extensão da classe *AWT* `Canvas`, fornecendo uma área retangular na tela para a renderização tridimensional de imagens e para a captura de eventos de entrada. Pode ser renderizado em modo monoscópico ou estereoscópico. No modo monoscópico o `Canvas3D` pode gerar três tipos de vista:

- `View.LEFT_EYE_VIEW`, a vista do olho esquerdo;
- `View.RIGHT_EYE_VIEW`, a vista do olho direito;
- `View.CYCLOPEAN_EYE_VIEW`, a vista correspondente ao ponto médio entre os olhos.

Estas políticas são selecionadas utilizando os métodos `setMonoscopicViewPolicy()` e `getMonoscopicViewPolicy()`.

Este projeto utilizou as duas primeiras políticas de geração da vista do modo monoscópico para obter as imagens esquerda e direita, que seriam posteriormente coloridas e somadas, resultando em um anaglifo. O modo estereoscópico pode ser utilizado em sistemas que possuam suporte de hardware para tal, como sistemas com óculos obturadores. (Sun Microsystems 2000).

O `Canvas3D` pode ser configurado para funcionar no modo ***on-screen***, no qual o seu *loop* de renderização é executado continuamente, renderizando a imagem na tela (desde que o mesmo esteja associado a um objeto `View` ativo), ou ***off-screen***, no qual a renderização é feita em um buffer de memória em resposta a uma chamada do método `renderOffscreenBuffer()`. (Sun Microsystems 2000). Esta capacidade é utilizada extensivamente neste projeto.

O buffer de um `Canvas3D` on-screen não pode ser acessado ou modificado, o que constituiu uma das principais dificuldades deste projeto, implicando em certas limitações na implementação realizada. (Sun Microsystems 2000).

O `Canvas3D` no modo *off-screen* não deve ser adicionado a nenhum `Container`. O Java 3D renderiza a cena no canvas em resposta ao método `renderOffScreenBuffer()`. O método `setOffScreenBuffer()` define um objeto do tipo `ImageComponent` que receberá a imagem renderizada para este objeto `Canvas3D`. (Sun Microsystems 2000).

O método `renderOffScreenBuffer()` é responsável pela renderização de um quadro no buffer de um canvas off-screen. A renderização só será efetuada se o objeto `Canvas3D` estiver associado a um objeto `View` ativo do universo em questão. Para uma aplicação saber quando a renderização está completa é preciso derivar a classe `Canvas3D` e sobrecarregar o método `postSwap()`, ou então utilizar o método `waitForOffScreenRendering()` (Sun Microsystems 2000).

Os métodos `setLeftManualEyeInImagePlate()` e `setRightManualEyeInImagePlate()` da classe `Canvas3D` definem a posição dos olhos esquerdo e direito, respectivamente, no modelo de visualização. Estes métodos são utilizados neste projeto para definir a separação interocular do observador. (Sun Microsystems 2000).

2.6.3 Shaders Programáveis no Java 3D: Hierarquia de Classes

A funcionalidade de shaders programáveis no Java 3D é fornecida pela hierarquia de classes ilustrada no diagrama da página a seguir, que mostra de maneira simplificada as principais classes do Java3D referentes a esta funcionalidade.

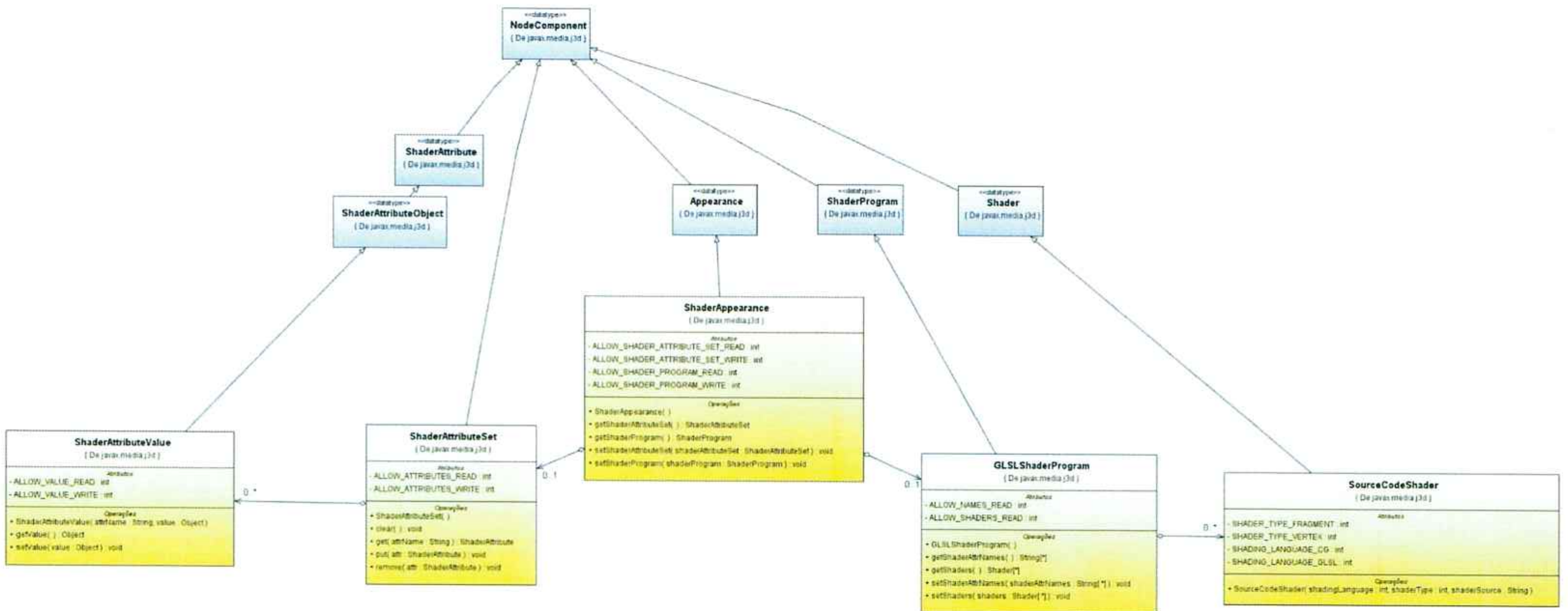

Figura 2-37: Hierarquia de classes que implementa a funcionalidade de shaders programáveis no Java3D (diagrama simplificado)

Uma breve descrição das classes é feita a seguir, para que se possa compreender melhor o funcionamento dos shaders programáveis dentro do Java3D.

SHADERAPPEARANCE

Esta classe define os atributos dos shaders programáveis que podem ser configurados como parte de um componente de um nó `Shape3D`. Ela acrescenta aos atributos já definidos pela classe `Appearance` os seguintes atributos (Sun Microsystems 2003):

- `shaderProgram`: objeto da classe `ShaderProgram` que contém um ou mais objetos contendo código-fonte de shaders em uma das linguagens suportadas pelo Java 3D (Java 3D 5.0: CG e GLSL).
- `shaderAttributeSet`: objeto do tipo `ShaderAttributeSet`, define os atributos que podem ser passados pela aplicação aos processadores de vértices e de fragmentos.

SHADERPROGRAM

Como o próprio nome indica, estas classes servem para manipular shaders escritos em uma das linguagens de programação para shaders suportada pelo Java3D, Cg ou GLSL. Objetos desta classe contém um ou mais objetos do tipo `SourceCodeShader`, que representam o código fonte dos shaders programáveis, em formato de texto. Esta é uma classe abstrata que é implementada por duas classes distintas, `CGShaderProgram` e `GLSLShaderProgram`. (Sun Microsystems 2003)

SOURCECODESHADER

Esta classe define objetos que contém o código fonte dos shaders programáveis, em formato de texto. Ela contém parâmetros para definir a linguagem do shader (CG ou GLSL) e o tipo de programa (vertex shader ou fragment shader). (Sun Microsystems 2003)

SHADERATTRIBUTESET

Um objeto do tipo `ShaderAttributeSet` fornece parâmetros uniformes aos shaders programáveis. Parâmetros uniformes são aqueles que são constantes

durante a renderização de uma primitiva geométrica. Seus valores podem mudar entre primitivas, porém são constantes para os vértices de uma mesma primitiva. Parâmetros uniformes incluem, por exemplo, matrizes de transformação geométrica, mapas de textura, luzes, tabelas de busca, etc. O objeto `ShaderAttributeSet` contém um conjunto de objetos do tipo `ShaderAttribute`. Cada objeto `ShaderAttribute` define o valor de um único parâmetro uniforme. (Sun Microsystems 2003)

ShaderAttributeValue

O objeto `ShaderAttributeValue` encapsula um parâmetro uniforme cujo valor é fornecido explicitamente, através dos campos `value` e `attrName`. Durante a renderização, o valor especificado em '`value`' é atribuído à variável '`attrName`' do shader programável. O campo `attrName` deve descrever o nome de um parâmetro uniforme válido no shader em que é utilizado, caso contrário o atributo será ignorado e um erro em tempo de execução poderá acontecer. O campo `value` deve ser uma instância de uma das classes permitidas, e deve corresponder ao tipo de dados da variável `attrName` no shader em que é utilizado. As classes permitidas são: `Integer`, `Float`, `Tuple{2,3,4}{i,f}`, `Matrix{3,4}f`. (Sun Microsystems 2003)

ShaderErrorListener

`ShaderErrorListener` é uma interface que monitora erros em shaders programáveis. Erros de compilação, ligação e em tempo de execução são reportados por esta interface. Uma classe que implemente esta interface deve implementar o método `errorOcurred(ShaderError error)`, que realiza o tratamento de erros. A classe é então associada a um objeto do tipo `VirtualUniverse` pelo método `addShaderErrorListener()`. (Sun Microsystems 2003)

ShaderError

Um objeto `ShaderError` serve como um container para os detalhes dos erros de compilação ou execução ocorridos em um shader. Este objeto é o parâmetro único do método `errorOcurred` da classe `ShaderErrorListener`. (Sun Microsystems 2003)

2.6.4 SHADERS PROGRAMÁVEIS NO JAVA 3D: EXEMPLO DE FUNCIONAMENTO

Esta seção visa esclarecer a sequência de operações necessárias para utilizar um shader programável dentro do Java 3D:

1. Escrever os programas *vertex shader* e *fragment shader* que substituirão a funcionalidade fixa do pipeline OpenGL, em uma das linguagens suportadas pelo Java 3D: Cg ou GLSL;
2. Criar os objetos `SourceCodeShader`, especificando:
   - O tipo de shader (vertex ou fragment);
   - A linguagem utilizada (Cg ou GLSL);
   - Uma string contendo o código-fonte do shader;
3. Criar um array de objetos do tipo `Shader`, que deve conter os objetos `SourceCodeShader` criados no passo anterior;
4. Criar o conjunto de objetos `ShaderAttributeValue` que definem os atributos passados aos shaders programáveis pela aplicação. Um objeto `ShaderAttributeValue` é composto de:
   - Uma string com o nome do atributo, exatamente como ele aparece no código-fonte do shader programável;
   - O valor do atributo, na forma de um objeto do tipo de dados desejado (por exemplo, `new Integer(1)` ao invés de `int 1`).
5. Criar um objeto `ShaderAttributeSet` que agrega o conjunto de atributos criados no passo anterior; Este objeto é composto por um ou mais objetos `ShaderAttributeValue`;
6. Criar um objeto `ShaderProgram`, que deve ser do mesmo tipo que a linguagem de programação utilizada no shader: `CGShaderProgram` para shaders em Cg ou `GLSLShaderProgram` para shaders em GLSL. Este objeto é então associado ao array de Shaders criados no passo 4, e aos nomes dos atributos criados no passo 5, através dos métodos `setShader()` e `setShaderAttrNames();`
7. Instanciar um objeto do tipo `ShaderAppearance` e associar a ele os objetos `ShaderProgram` e `ShaderAttributeSet` criados nos passos 6 e 7;

8. Associar o objeto `ShaderAppearance` a um nó `Shape3D` do grafo de cena. Após este passo o objeto `Shape3D` já será renderizado com a aplicação do shader.
9. *Opcional:* Instanciar um objeto que implemente a interface `ShaderErrorListener` e associá-lo a `VirtualUniverse`. Embora seja opcional, este passo é altamente recomendável, pois possibilita a captura de qualquer erro de compilação, ligação ou execução ocorrido no shader, com uma mensagem descritiva que facilita o rastreamento de erros no programa. O seguinte trecho de código exemplifica esta operação:

```
myUniverse.addShaderErrorListener(new ShaderErrorListener() {
     public void errorOccurred(ShaderError error) {
         error.printVerbose();
     }
});
```

Fragmento de Código 2-1: Acrescentando um ShaderErrorListener a um VirtualUniverse

Pelo processo descrito, percebe-se que a associação de um shader é feita no nível das folhas do sub-grafo de conteúdo, aplicando-se apenas ao nó `Shape3D` associado ao objeto `ShaderAppearance` criado no processo. Portanto, para aplicar o mesmo shader a todos os nós `Shape3D` de um grafo de cena, como é o caso na renderização do anaglifo, devemos instanciar diversos objetos `ShaderAppearance` com o mesmo código-fonte do shader, e o mesmo conjunto de atributos `ShaderAttributeSet`.

3. METODOLOGIA

Esta seção descreve os métodos de pesquisa, projeto e desenvolvimento utilizados neste projeto.

3.1 METODOLOGIA DE PESQUISA

A primeira parte do projeto consistiu no estudo dos principais tópicos envolvidos neste projeto, que são descritos abaixo juntamente com a citação das principais referências utilizadas:

- RA e RAE: principais conceitos e definições, aplicações existentes, problemas enfrentados, etc. (R. T. Azuma 1997), (Hoff, Nguyen e Lyon 1996), (Milgram, et al. 1994), (Raskar 1998), (Tori, Kirner e Siscoutto 2006);
- Estereoscopia: princípios envolvidos na percepção de profundidade no ser humano, formas de reprodução artificial destes princípios. (ALBUQUERQUE 2006), (Raposo, et al. 2004), (Sousa 2006), (TOMMASELLI 2006), (StereoGraphics Corporation 1997);
- Estereoscopia com anaglifos: conceito de anaglifo, método de obtenção de um anaglifo, modos possíveis para reprodução de um anaglifo (puro, escala de cinza, em cores, etc.). (Raposo, et al. 2004), (3dtv.at 2005);
- API Java 3D: conceitos principais, grafos de cena, modos de renderização, hierarquia de classes, estudo de exemplos, estudo do suporte a visão estéreo. (Sun Microsystems 2001), (Sun Microsystems 2000), (Sun Microsystems 2003), (Manssour 2003);
- OpenGL: pipeline gráfico, funcionalidade fixa, shaders programáveis. (Segal e Akeley 2006), (Fernandes 2006), (John Kessenich 2006);
- Shaders programáveis no Java 3D: hierarquia de classes e seu funcionamento, interações entre as classes, estudo de exemplos. É importante notar que a documentação oficial das classes, disponível em (Sun Microsystems 2003) está incompleta e/ou desatualizada; o conhecimento necessário sobre este tópico foi obtido principalmente

através do estudo de exemplos e realização de implementações de teste. (Sun Microsystems 2003); (Rushforth e Yang 2005).

A implementação do software foi iniciada após extensa revisão bibliográfica e conceituação teórica inicial. Mesmo assim, como não se possuía nenhum conhecimento prévio da tecnologia abordada, boa parte do trabalho de pesquisa referente aos assuntos técnicos foi realizado durante o desenvolvimento. Esta pesquisa foi feita à medida em que a aquisição de novos conhecimentos sobre estes assuntos se fez necessária para avançar no desenvolvimento do sistema.

3.2 METODOLOGIA DE PROJETO E DESENVOLVIMENTO

O programa desenvolvido foi baseado nos resultados obtidos por Haseyama (2007), projeto do qual participei em suas fases iniciais. A implementação de Haseyama (2007) foi bem-sucedida em obter um sistema de projeção estereoscópica com anaglifos, porém a baixa taxa de quadros obtida limitou a aplicabilidade do sistema. Partindo da implementação obtida neste trabalho, foi iniciado o desenvolvimento de uma nova implementação com o objetivo de aprimorar o desempenho obtido pelo sistema anterior no quesito de quadros por segundo.

O desenvolvimento do software seguiu uma metodologia orientada a protótipos, com a realização de iterações de desenvolvimento e estruturação de código resultando em acréscimos incrementais de funcionalidade. O uso de uma abordagem incremental, pouco estruturada, se justificou devido principalmente aos seguintes aspectos, em ordem de prioridade:

- Equipe de desenvolvimento constituída de apenas um desenvolvedor;
- Simplicidade estrutural do modelo previsto;
- Conhecimento limitado da tecnologia utilizada, com a obtenção de boa parte do conhecimento técnico realizada conforme a demanda indicada pelo desenvolvimento de novas funcionalidades;
- Tecnologia com documentação incompleta (no caso do uso de shaders no Java3D), exigindo o estudo de seu funcionamento através de exemplos e implementações de teste;
- Projeto de caráter inovador;

Dadas estas características, a definição prévia de um modelo estrutural através de diagramas e outros documentos de projeto mostrou-se desnecessária e até mesmo contra-produtiva devido à alta probabilidade de mudanças ao longo do desenvolvimento. Optou-se então pela utilização de uma abordagem direta (*hands-on approach*) para o desenvolvimento do software. Esta abordagem assemelha-se bastante à metodologia XP (Extreme Programming) definida em (Wells 1999).

Entretanto, este tipo de abordagem apresenta sérias desvantagens que devem ser compensadas, como:

- Produção do software de forma pouco estruturada, implicando em arquiteturas pouco robustas e sujeitas a falhas estruturais;
- A geração de documentação não é uma preocupação principal da metodologia, dificultando o reuso do software por terceiros;
- O código-fonte gerado é potencialmente mais desorganizado e difícil de ser compreendido, com uma tendência ao surgimento de defeitos.

Esforços foram feitos ao longo do desenvolvimento para mitigar as desvantagens introduzidas pelo uso de uma metodologia de desenvolvimento pouco estruturada. Em primeiro lugar, tomou-se o cuidado de introduzir iterações de organização e estruturação de código entre as iterações de prototipação, resultando em um programa mais limpo, estruturado e legível. Finalmente, a introdução de comentários no código-fonte, esclarecendo o funcionamento das classes criadas e suas principais operações facilitou a geração posterior da documentação.

O princípio fundamental de desenvolvimento utilizado neste projeto foi a construção de protótipos de complexidade crescente através de uma seqüência de iterações; cada iteração foi realizada com o propósito de produzir um novo resultado tangível para o programa, isto é, introduzir novas funcionalidades. Este princípio apresenta como vantagem a obtenção rápida de resultados em cada iteração, possibilitando a avaliação imediata do grau de sucesso de uma certa abordagem na resolução de um problema.

A implementação inicial baseou-se na estrutura de classes definida em (Haseyama 2007). Esta implementação serviu como referência estrutural para as primeiras iterações de desenvolvimento; entretanto, a versão final do software

tornou-se completamente distinta desta, tendo mantido inalterados poucos aspectos desta; a saber, apenas uma classe para a exibição de uma imagem em um JPanel, e um algoritmo que efetua a soma dos pixels de duas imagens para compor uma terceira.

A criação dos diagramas que descrevem a estrutura de classes do programa foi realizada posteriormente ao desenvolvimento, como forma de documentar a implementação realizada.

A descrição de cada iteração de desenvolvimento é feita na seção 5.

4. ESPECIFICAÇÃO DO PROJETO

Este capítulo trata dos requisitos técnicos necessários para a execução do aplicativo, bem como de algumas características relevantes a respeito da visualização de anaglifos. Também fornece detalhes sobre a funcionalidade implementada pelo programa, o diagrama do processo de produção de um anaglifo, e o diagrama de classes do software desenvolvido.

4.1 REQUISITOS DE HARDWARE

O computador que executará o sistema deve possuir os seguintes requisitos mínimos de hardware:

- Processador de 500 MHz;
- 256 MB de memória RAM;
- Placa de vídeo com suporte a OpenGL 2.0 e shaders programáveis, com no mínimo 32 MB de memória;

Os requisitos recomendados para a execução do sistema são:

- Processador de 1 GHz ou superior;
- 512 MB ou mais de memória RAM;
- Placa de vídeo com suporte a OpenGL 2.0 e shaders programáveis, com 128 MB de memória;

O sistema pode utilizar como dispositivo de saída um monitor comum ou um projetor do tipo *datashow*, dependendo dos requisitos da aplicação final.

Para visualização do efeito estereoscópico os usuários deverão utilizar óculos com lentes coloridas vermelho/verde ou vermelho/azul como os da figura a seguir. A lente vermelha corresponde ao olho esquerdo; a lente verde/azul corresponde ao olho direito.

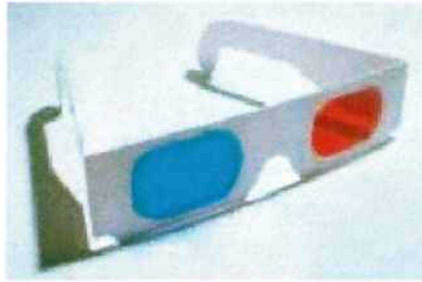

Figura 4-1- Óculos utilizados para visão estéreo com anaglifo. (Tori, Kirner e Siscoutto 2006)

4.2 REQUISITOS DE SOFTWARE

O sistema no qual o programa será executado deve ter o Java Runtime Environment (JRE) versão 1.6.0 ou superior e o Java 3D versão 1.5.0 ou superior instalados.

4.3 CONSIDERAÇÕES SOBRE OS USUÁRIOS DO SISTEMA

A utilização dos óculos anaglíficos por períodos prolongados pode provocar sensações de desconforto como náuseas, mal-estar e dores de cabeça, embora nenhuma destas condições tenha sido observada pelos desenvolvedores e usuários de teste durante o desenvolvimento do sistema.

Uma condição peculiar experimentada pelos usuários foi a persistência da coloração dos filtros na visão do usuário ao retirar os óculos, mesmo para curtos períodos de visualização. Após retirar os óculos, a visão do olho esquerdo apresentou tendência para o azul, enquanto a visão do olho direito apresentou tendência para o vermelho. O fato curioso a este respeito é que a persistência de cores na visão apresentou inversão em relação à posição das lentes nos óculos, isto é, o olho cujo filtro era azul apresentou tendência para o vermelho, e vice-versa.

Este fato pode ser verificado após um curto período de visualização com os óculos, retirando-se os mesmos e fechando cada um dos olhos alternadamente. Desta forma pode-se observar a predominância de uma das cores na visão de cada olho. Nenhuma investigação foi realizada neste projeto a respeito deste fenômeno; não se sabem as causas e possíveis efeitos a longo prazo do mesmo.

Usuários com problemas de visão podem ter dificuldades ou até mesmo não perceberem o efeito estereoscópico. Além disso, mesmo em usuários com visão

perfeita o efeito pode não ser percebido, como foi constatado com alguns dos usuários de teste.

4.4 Funcionalidade Implementada

O sistema desenvolvido implementa a seguinte funcionalidade:

- Construção de um grafo de cena do Java 3D com animação e interatividade;
- Renderização contínua da cena com o efeito de estereoscopia por anaglifo;
- Possibilidade de alternar entre cinco diferentes modos de produção do anaglifo;
- Possibilidade de ajuste da separação interocular do observador;
- Possibilidade de pausar e resumir a animação e interatividade da vista exibida;

4.5 Diagrama de Blocos do Processo de Produção de um Anaglifo

O diagrama a seguir, extraído de (Haseyama 2007), ilustra o processo de produção de um anaglifo num nível abstrato:

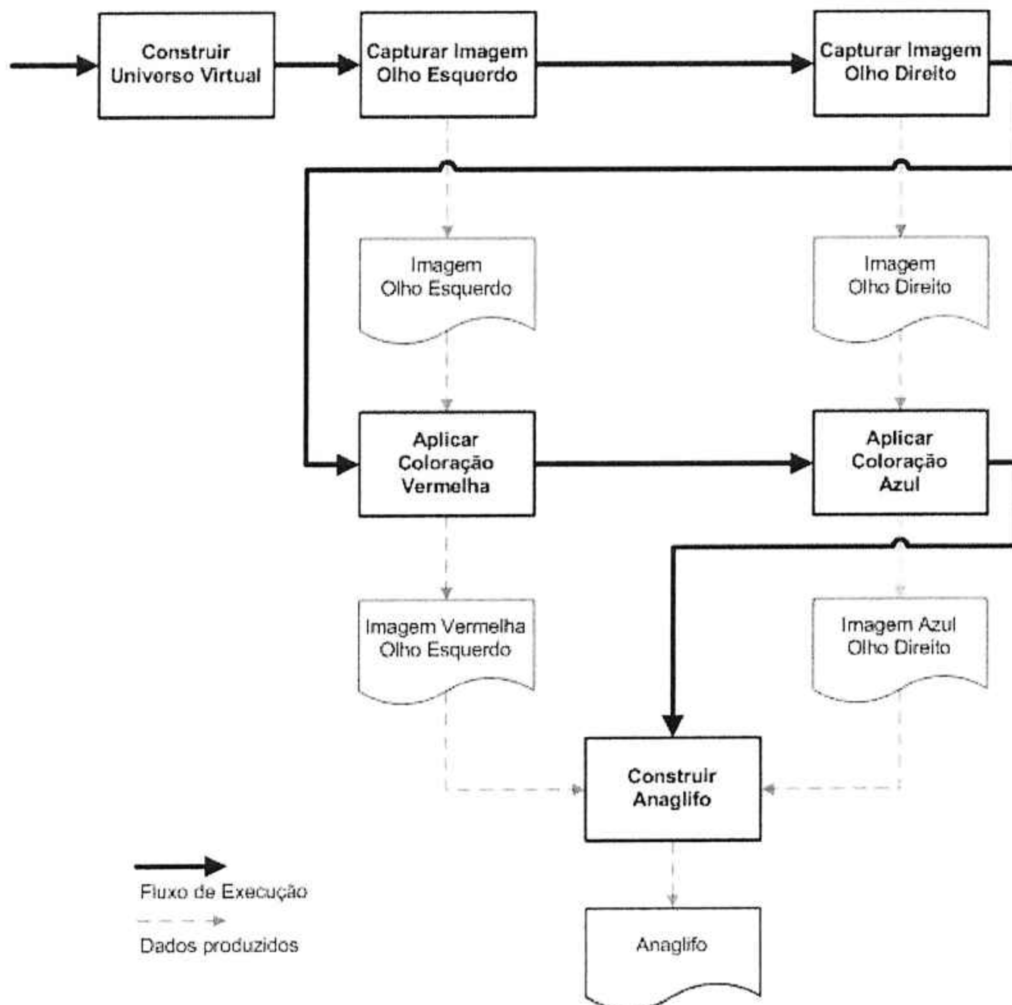

Figura 4-2: Diagrama de blocos para a produção de um anaglifo

4.6 DIAGRAMA DE CLASSES DO SISTEMA

O diagrama da figura 4-3 representa a estrutura de classes do sistema:

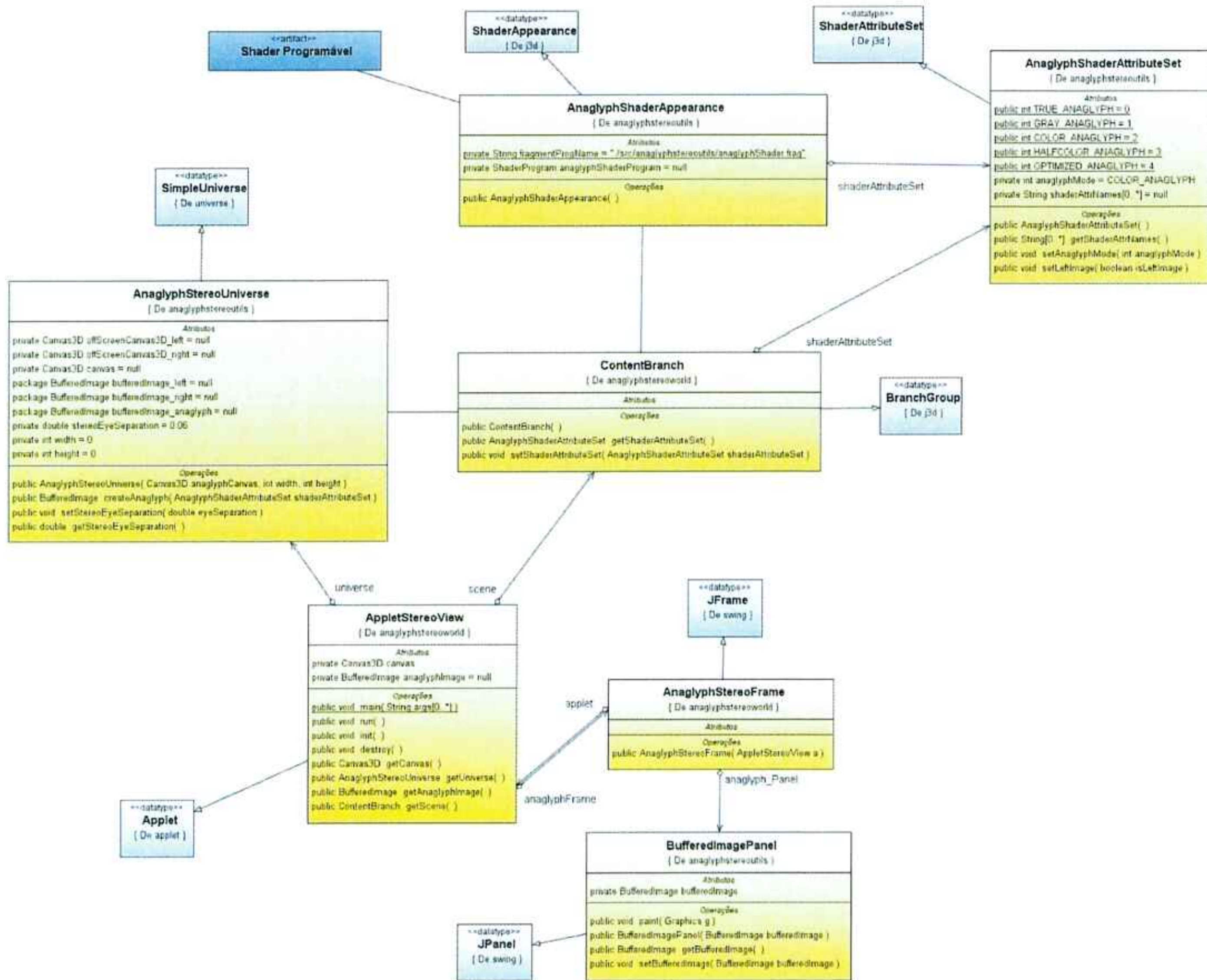

Figura 4-3: Diagrama de classes do projeto principal

O sistema está dividido em dois pacotes: `anaglyphstereoutils`, que contém as classes utilitárias do sistema, e `anaglyphstereoworld`, que contém as classes da aplicação principal, a interface gráfica, e o sub-grafo de conteúdo do Java 3D. Uma descrição breve das classes é dada a seguir:

AnaglyphStereoUniverse

Esta é a classe principal do pacote `anaglyphstereoutils`. Ela é responsável por criar o sub-grafo de visualização do grafo de cena e os objetos `Canvas3D` que são utilizados para renderizar as imagens esquerda e direita do anaglifo. Além disso, a captura das imagens do par estereoscópico e a soma subseqüente das imagens para compor o anaglifo é feita por esta classe.

AnaglyphShaderAppearance

Esta classe é responsável pelas operações necessárias para carregar um shader programável no pipeline do OpenGL. É a classe que implementa a sequência de operações descrita na seção 2.6.4.

AnaglyphShaderAttributeSet

Esta classe contém o conjunto de atributos passados da aplicação para o shader, bem como os métodos utilizados pela aplicação para atualizar estes atributos.

O Shader Programável

O shader programável é um programa à parte, escrito na linguagem de shaders do OpenGL, o GLSL. Este programa é contido em um arquivo de texto comum, acessado pela classe `AnaglyphShaderAppearance`. O shader programável é responsável pelas operações de coloração das imagens esquerda e direita do anaglifo.

BufferedImagePanel

A última classe do pacote `anaglyphstereoutils` é utilizada para exibir uma imagem na tela. Esta classe foi extraída de (Haseyama 2007).

ContentBranch

Esta classe está contida no pacote `anaglyphstereoworld`, e implementa o sub-grafo de conteúdo do grafo de cena do Java 3D. Este sub-grafo pode ser composto por quaisquer objetos do Java 3D. Dentro da proposta deste projeto, o sub-grafo utilizado deve possuir capacidades de animação e interação.

AppletStereoView

Esta é a classe de aplicação do pacote `anaglyphstereoworld`, responsável pela inicialização de objetos e entrada da função principal do programa. Ela pode ser executada como um applet em uma janela de navegador ou como um aplicativo *stand-alone*.

AnaglyphStereoFrame

Esta é a classe que implementa a interface gráfica do sistema, contendo os controles de execução e a área retangular da tela onde o anaglifo é renderizado.

5. PROJETO E IMPLEMENTAÇÃO

Este capítulo apresenta os aspectos principais da implementação do sistema. Inicialmente é feita uma descrição de cada iteração de desenvolvimento realizada, e finalmente os detalhes de implementação de cada classe são fornecidos.

5.1 DESCRIÇÃO DAS ITERAÇÕES DE DESENVOLVIMENTO

5.1.1 CONSTRUÇÃO DE UM GRAFO DE CENA SIMPLES NO JAVA 3D

A finalidade desta iteração foi a familiarização com a hierarquia de classes e modo de operação do Java 3D. Foi criado um grafo de cena contendo apenas um objeto do tipo `Box` no formato de um cubo. O código pôde então ser modificado diversas vezes para avaliar os efeitos da manipulação de parâmetros como: a transformada de posição, rotação e escala do cubo, a transformada do ponto de vista, alterações no objeto `Appearance` deste cubo, aplicação de texturas, backgrounds, etc. A figura 5-1 mostra uma renderização deste grafo de cena:

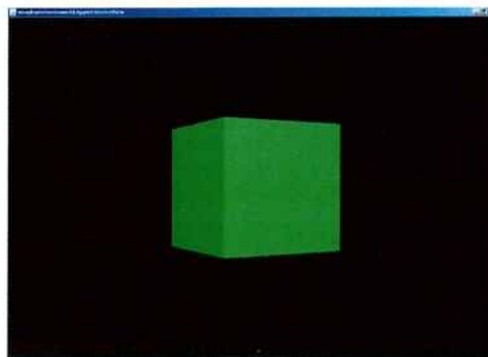

Figura 5-1: Grafo de cena contendo apenas um cubo

5.1.2 CODIFICAÇÃO E APLICAÇÃO DE UM SHADER PROGRAMÁVEL SIMPLES

Nesta iteração foi escrito um fragment shader simples, que altera a cor dos pixels do objeto para uma única cor. A principal dificuldade nesta fase do projeto foi o acesso a informações precisas sobre o funcionamento de shaders programáveis no Java 3D, já que a documentação contida nas páginas oficiais está incompleta e/ou desatualizada. Para superar este problema, foi realizado o estudo de implementações de aplicativos Java 3D já existentes, e que utilizam shaders de uma

forma análoga, com a passagem de parâmetros entre o Java 3D e os shaders. A figura 5-2 mostra uma renderização deste grafo de cena:

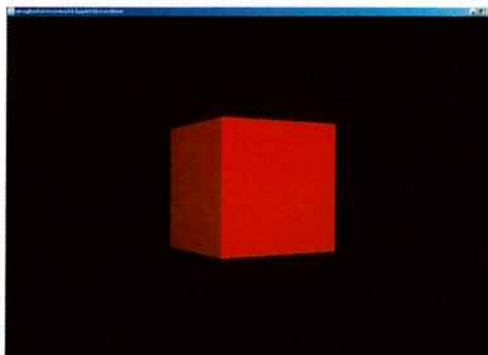

Figura 5-2: Renderização do objeto com um fragment shader simples associado

5.1.3 ALTERAÇÃO DO SHADER PARA A OBTENÇÃO DAS CORES DO ANAGLIFO

Esta iteração consistiu em alterar o shader escrito para utilizar as componentes de cor nas proporções corretas para a produção do anaglifo, como descrito em (3dtv.at 2005). Foram utilizados dois shaders distintos, um para a cor vermelha e outro para a cor azul. A figura 5-3 mostra as renderizações obtidas após esta alteração:

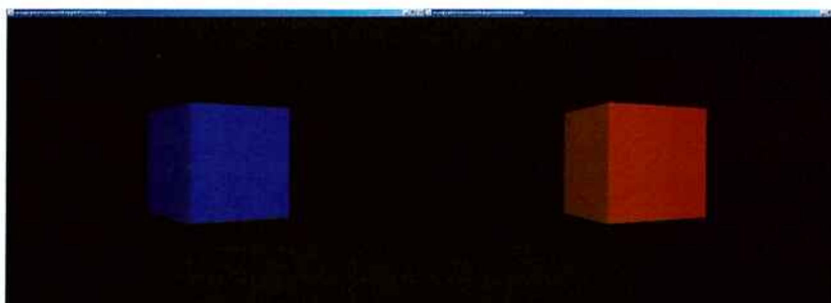

Figura 5-3: Aplicação de dois fragment shaders, um para a cor azul e outro para a cor vermelha do anaglifo

5.1.4 APLICAÇÃO DE DOIS SHADERS E SOMA DAS IMAGENS RESULTANTES

Esta iteração concentrou boa parte do esforço de desenvolvimento do programa. O principal desafio encontrado nessa etapa foi a implementação do método de renderização seqüencial das imagens do par estereoscópico. Inicialmente tentou-se estender a classe `Canvas3D` para utilizar outros modos de renderização: modo imediato ou modo misto. Esta abordagem não obteve sucesso, já que a renderização no modo misto ocorre em uma só passagem (assim como no modo automático), não permitindo a captura de duas imagens do mesmo quadro, e

a renderização no modo imediato foi descartada pela complexidade excessiva e incerteza de sua capacidade em fornecer a funcionalidade desejada.

A abordagem que obteve sucesso utilizou-se da instanciação de dois objetos `Canvas3D` no modo off-screen, cada um com uma política diferente de geração da vista. O canvas esquerdo utiliza a política `View.LEFT_EYE_VIEW`, enquanto o canvas direito utiliza a política `View.RIGHT_EYE_VIEW`. Estas políticas fazem parte do suporte a estéreo nativo do Java 3D, e geram automaticamente as vistas esquerda e direita do observador.

Para a coloração das imagens do par estereoscópico foram utilizados dois fragment shaders distintos, contidos nos objetos `shaderProgramRed` e `shaderProgramBlue`. O método `createAnaglyph()` alternava em tempo de execução o shader programável associado ao objeto `ShaderAppearance` do cubo. Desta forma, foi possível realizar a renderização dos canvas off-screen utilizando um shader diferente para cada um deles. A soma das imagens contidas em seus buffers fornecia um anaglifo.

Após estas operações restava ainda o problema da exibição do anaglifo em tela. Como um objeto `Canvas3D` no modo on-screen não permite a manipulação de seu buffer, e um objeto `Canvas3D` no modo off-screen não pode ser exibido em tela, foi necessário recorrer a um objeto da classe `BufferedImagePanel`, definida em (Haseyama 2007). Esta classe é capaz de exibir na tela uma imagem contida em um objeto `BufferedImage`.

A imagem abaixo ilustra a renderização obtida nesta etapa:

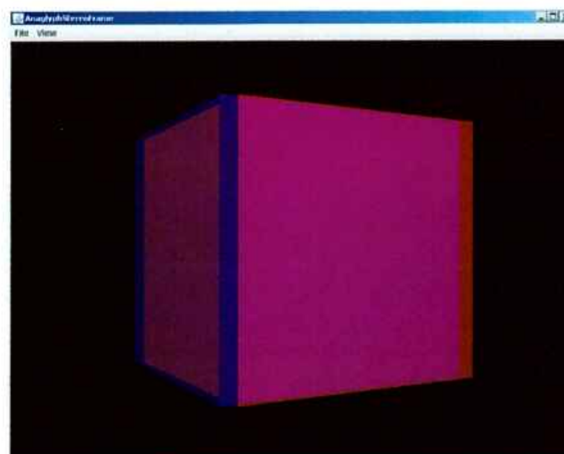

Figura 5-4: Renderização de um cubo com anaglifo

5.1.5 CONTROLE DO MODO DE PRODUÇÃO DO ANAGLIFO

Após obter uma renderização bem-sucedida do anaglifo, os modos de produção citados no item 2.4.3 foram implementados de acordo com as fórmulas extraídas de (3dtv.at 2005). Os modos são: puro, escala de cinza, em cores, semi-colorido e otimizado. Para a implementação dos modos foi criada uma estrutura de controle no shader programável, controlada pelo parâmetro `anaglyphMode` passado pela aplicação para o shader. A figura abaixo mostra os diferentes modos de anaglifo em funcionamento:

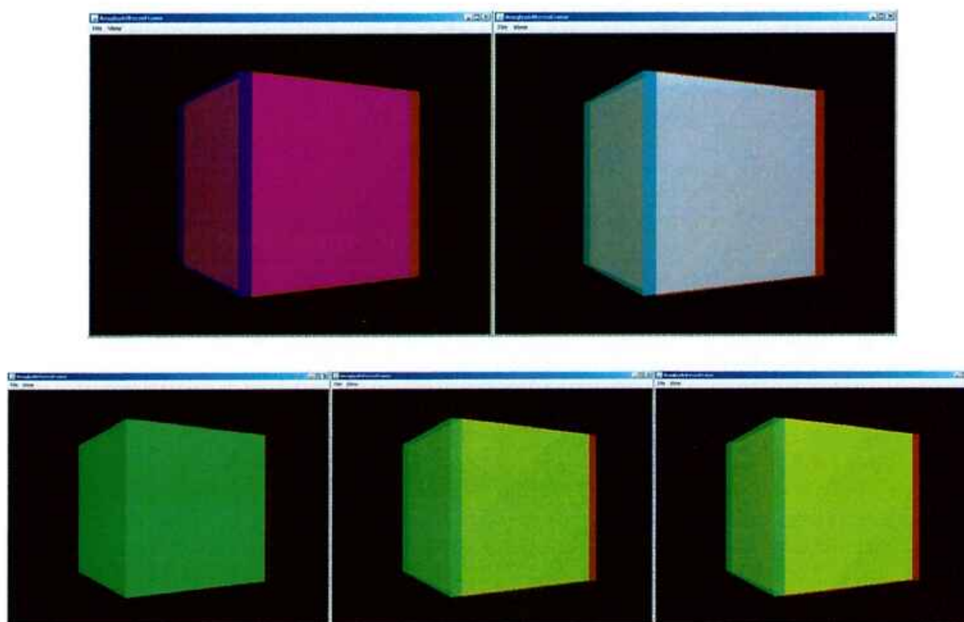

Figura 5-5: Renderização do anaglifo em diferentes modos: a) puro; b) escala de cinza; c) em cores; d) semi colorido; e) otimizado

5.1.6 CONTROLE DA SEPARAÇÃO INTEROCULAR DO OBSERVADOR

Nesta iteração foi introduzido o controle da separação interocular do observador, através da manipulação de dois objetos `Point3D` que representam as posições dos olhos do observador. Estes objetos são acessados através do objeto `Viewer` que contém um objeto `PhysicalBody`. O objeto `PhysicalBody` descreve as propriedades físicas do observador, como por exemplo as posições dos olhos. Estes objetos são criados automaticamente pela construtora da classe `SimpleUniverse`. A figura 5–6 ilustra a mesma cena com separações interoculares diferentes:

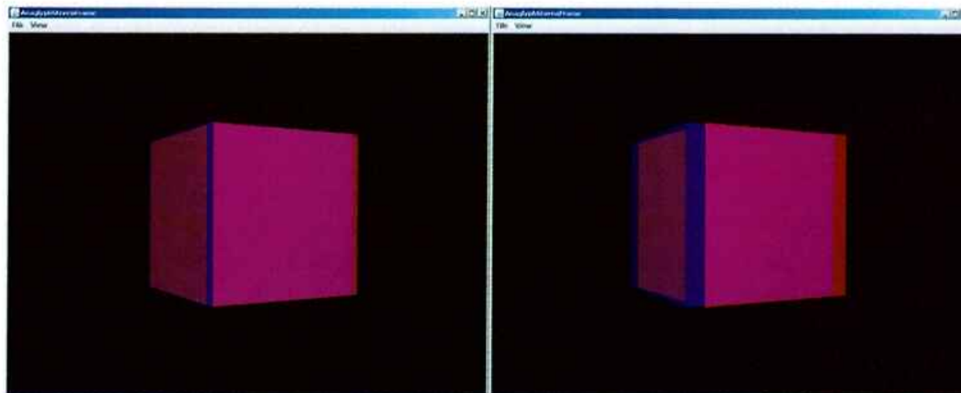

Figura 5-6: Cena com separações interoculares diferentes: a) separação menor; b) separação maior

O ajuste da separação interocular é uma funcionalidade interessante, pois os usuários do sistema podem ter características físicas bastante distintas entre si; com esta funcionalidade o sistema pode ser adaptado a diferentes usuários para obter o melhor resultado em cada caso.

5.1.7 INTRODUÇÃO DE MÚLTIPLOS OBJETOS NO GRAFO DE CENA

Os resultados obtidos até então foram restritos à aplicação do efeito de anaglifo em um único objeto do grafo de cena. O objetivo desta etapa foi estender a funcionalidade implementada para suportar a aplicação do efeito anaglífico em um grafo de cena contendo múltiplos objetos.

O suporte a mais de um objeto exigiu a associação de um objeto `ShaderAppearance` para cada `Shape3D` instanciado, para que cada objeto do grafo de cena possa ter seus atributos de cor, reflexão, etc. ajustados separadamente. Até então, o objeto `ShaderAppearance` continha dois shaders programáveis, um para a cor vermelha e outro para a cor azul. A aplicação alternava entre estes programas em tempo de execução para fazer a captura das imagens esquerda e direita, utilizando para isso um método da classe `ShaderAppearance`.

Com o uso de mais de um objeto `ShaderAppearance`, esta forma de controle se mostrou pouco robusta e propensa a falhas. Passou-se então a utilizar um único shader programável que integrava os algoritmos para produção tanto da imagem vermelha quanto da imagem azul/verde. A seleção entre as imagens pôde ser feita com a introdução da variável booleana `isLeftImage` no conjunto de parâmetros do shader.

Neste ponto, foi necessário decidir entre utilizar apenas um objeto `ShaderAttributeset` global para atualizar os atributos de todos os shaders, ou utilizar múltiplos destes objetos e construir um vetor deles para controlar a atualização dos parâmetros em todos os shaders antes da renderização de cada imagem do par. Optou-se pela primeira forma por sua maior simplicidade, porém sem nenhum prejuízo para a funcionalidade proposta. O programa pode ser facilmente estendido para suportar a segunda forma. Esta segunda forma permite que cada shader defina efeitos de renderização diferentes para o seu objeto `Shape3D`, em adição ao efeito do anaglifo.

A figura abaixo mostra uma renderização da cena com múltiplos objetos:

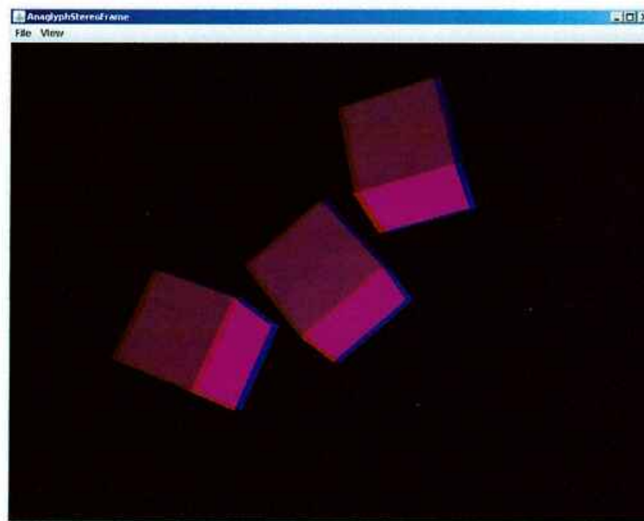

Figura 5-7: Renderização do grafo de cena com múltiplos objetos

5.1.8 SUPORTE À RENDERIZAÇÃO DE CENAS DINÂMICAS

O próximo desenvolvimento necessário foi a introdução de animações e interatividade no grafo de cena, pois até então todos os objetos do grafo de cena eram estáticos. Foi preciso implementar o suporte à atualização dinâmica da cena renderizada, possibilitando o uso de animações e interatividade.

Para conseguir a renderização contínua, foi implementada uma Thread na classe principal da aplicação, que pode ser executada concorrentemente com as outras threads do Java 3D, como o loop de renderização e o controle de *Behaviors*, sem interferir nestas operações.

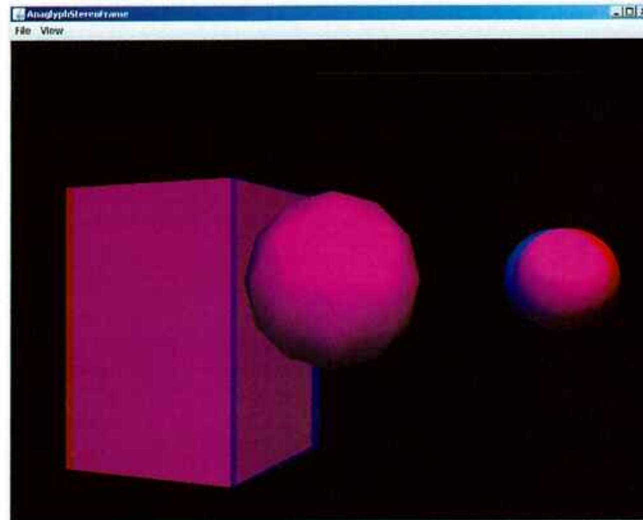

Figura 5-8: Grafo de cena com múltiplos objetos animados e com interatividade

Nesta etapa de desenvolvimento deparou-se com um problema relacionado à forma como o Java 3D trata a interatividade. A captura de eventos de entrada pelo aplicativo Java 3D é feita sobre a área da tela coberta pelo Canvas on-screen correspondente. Pelos motivos citados na seção 5.1.4, o anaglifo não é exibido por um `Canvas3D` e sim pela classe `BufferedImagePanel`. Portanto, o tratamento dos eventos de entrada exigiu que a aplicação criasse duas janelas, uma contendo o anaglifo e outra contendo o próprio Canvas on-screen que captura os eventos de entrada. Esta limitação não constitui um impedimento grave para a demonstração do efeito estereoscópico.

Uma solução para este problema é certamente possível, realizando a captura dos eventos de entrada no próprio objeto `BufferedImagePanel` e passando estes eventos para o Java 3D. Entretanto, não foi possível buscar uma solução neste modelo durante o projeto. A presença desta e de outras possibilidades de aprimoramento do sistema são discutidas na seção 7.

5.1.9 IMPLEMENTAÇÃO DE UMA INTERFACE GRÁFICA SIMPLES

Nesta iteração foram acrescentados controles gráficos para a alteração dos modos de produção do anaglifo, e ajuste da separação interocular do observador, que até então só podiam ser feitos por atalhos de teclado. Também foi introduzida uma funcionalidade para pausar e resumir a animação da cena. A figura 5-9 ilustra esta interface:

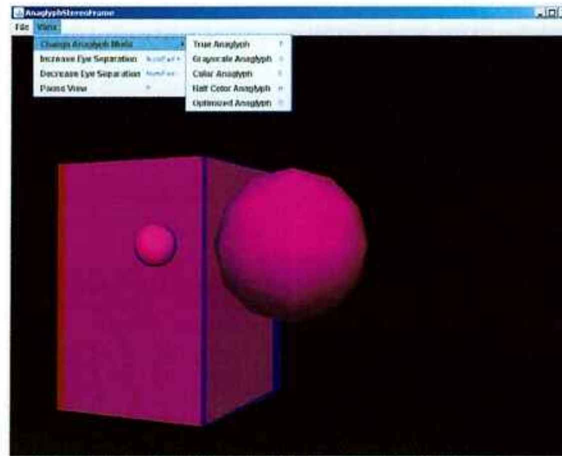

Figura 5-9: Interface gráfica do aplicativo

5.1.10 CRIAÇÃO DE UM FRAMEWORK E OTIMIZAÇÃO DO CÓDIGO GERADO

Esta iteração serviu para organizar o código gerado em um framework, separando o pacote de classes utilitárias das classes da aplicação principal. As classes foram separadas conforme as suas responsabilidades, visando facilitar o reuso do código.

Além disso, diversas otimizações foram introduzidas nesta etapa, obtendo um ganho considerável de desempenho no uso de memória e desempenho em taxas de quadro do sistema. Embora o ganho tenha sido perceptível, não foram obtidos valores para mensurar o quanto o código foi otimizado.

5.2 DESCRIÇÃO DAS CLASSES CONTIDAS NA VERSÃO FINAL DO SOFTWARE

5.2.1 CLASSE ANAGLYPHSTEREOUNIVERSE

Esta classe estende a classe `SimpleUniverse` do pacote de classes utilitárias do Java 3D. É a classe principal do pacote `anaglyphstereoutils`, responsável pelas seguintes operações:

- Construção do sub-grafo de visualização do grafo de cena;
- Criação de dois objetos `Canvas3D` no modo offscreen e associação destes ao objeto `View` do sub-grafo de visualização;
- Criação de uma renderização em anaglifo do grafo de cena associado a este objeto, através do método `createAnaglyph`;
- Ajuste da separação interocular do observador, pelo método `setStereoEyeSeparation`.

O MÉTODO CREATEANAGLYPH

O método `createAnaglyph` faz a captura das imagens esquerda e direita já coloridas pelo *fragment shader* e efetua a soma destas duas imagens em uma terceira, gerando o anaglifo.

Este método deve interromper temporariamente a renderização e o agendamento de *Behaviors* da cena. Esta operação é necessária para que a captura das duas imagens contidas nos canvas off-screen correspondam ao mesmo quadro, já que a renderização de um `Canvas3D` no modo offscreen é assíncrona por definição.

Como contra-exemplo para justificar este procedimento, imaginemos que o processo de renderização não seja interrompido: durante a captura de ambas as imagens, todos os procedimentos de animação de objetos e interatividade da cena estarão executando normalmente; isto fará com que alguns objetos da cena possivelmente se movam após a captura da imagem esquerda, porém antes da renderização da imagem direita, fazendo com que as duas imagens não coincidam.

Este fato é um dos principais limitantes no desempenho do sistema em taxa de quadros, já que cada quadro deve incluir duas operações de renderização completas, uma para cada vista.

Após interromper o loop de renderização e animação da cena, a próxima operação executada é a renderização dos `Canvas3D` off-screen e a captura de seus buffers de imagem. Esta operação é análoga para os dois olhos; o estado do parâmetro `leftImage`, contido no objeto `shaderAttributeset`, indica ao shader programável se a imagem sendo renderizada corresponde ao olho esquerdo (vermelho) ou direito (azul/verde). A coloração das imagens é realizada automaticamente durante a renderização da cena pelo shader programável, que executa no pipeline gráfico da placa de vídeo.

Para garantir que a renderização do buffer esteja completa antes de capturarmos a imagem, o método `waitForOffScreenRendering()` deve ser chamado. Caso contrário, há o risco de a imagem ser capturada com conteúdo nulo, gerando uma exceção em outra parte do código. Como já foi mencionado, este é um dos limitantes do desempenho do sistema.

Depois de capturar as duas imagens do par estereoscópico com as cores já ajustadas, o método realiza a soma das duas imagens capturadas, gravando o resultado em um buffer à parte. Esta operação consiste de um OU lógico entre os pixels das imagens esquerda e direita.

Este é outro fator limitante do desempenho do sistema em taxa de quadros, já que o número de passos do loop é igual ao número de pixels da imagem. Em uma resolução de 800 x 600 pixels, isto representa um total de 480,000 iterações. Uma quantificação do esforço computacional necessário neste loop é indicada na seção 6 - Resultados.

A última operação deste método antes de retornar um objeto `BufferedImage` contendo o anaglifo é a retomada do loop de renderização e do processamento de *Behaviors*, através das chamadas a `View().startBehaviorScheduler()` e `canvas.startRenderer()`

AJUSTE DA SEPARAÇÃO INTEROCULAR DO OBSERVADOR

Outra operação importante fornecida por esta classe é o ajuste da separação interocular do observador, através do método `setStereoEyeSeparation`, que ajusta as posições dos olhos esquerdo e direito no modelo de visualização do Java 3D.

5.2.2 CLASSE ANAGLYPHSHADERAPPEARANCE

Esta classe estende a classe `ShaderAppearance` do Java 3D. Enquanto a classe `ShaderAppearance` apenas fornece as funcionalidades necessárias para o carregamento de um shader programável, deixando a execução destas operações sob responsabilidade da aplicação, a classe `AnaglyphShaderAppearance` executa toda a sequência de operações necessárias para o carregamento do shader programável no pipeline gráfico. Isto simplifica a tarefa da aplicação, que deve apenas criar uma instância desta classe para cada objeto `Shape3D` no grafo de cena. A sequência de operações para utilização de shaders no Java 3D, executada por esta classe, é descrita na seção 2.6.4.

Um detalhe importante desta classe é que todos os objetos `AnaglyphShaderAppearance` do grafo de cena devem fazer referência ao mesmo objeto `AnaglyphShaderAttributeSet`, para que o algoritmo de criação do anaglifo funcione corretamente. Esta característica é explicada na seção 5.1.7 e não implica em redução da funcionalidade proposta. Contudo, esta condição pode ser facilmente contornada com poucas modificações no programa, caso seja desejável aplicar efeitos diversos a cada nó `Shape3D` além do efeito anaglífico.

5.2.3 CLASSE ANAGLYPHSHADERATTRIBUTESET

Esta classe estende a classe `ShaderAttributeSet` do Java 3D com a seguinte funcionalidade:

- Instanciação dos objetos `ShaderAttributeValue` que contém o nome e valor de cada variável utilizada no shader programável;
- Um método para alterar o modo de construção do anaglifo;

- Um método para alterar o valor de uma variável booleana que indica ao shader programável qual é a imagem sendo renderizada, esquerda ou direita.

Os objetos `ShaderAttributeValue` instanciados por esta classe são utilizados pelo shader programável para a construção do anaglifo. Estes atributos são: `anaglyphMode`, um inteiro que indica o modo de anaglifo sendo utilizado, e `leftImage`, uma variável booleana que indica se a imagem sendo renderizada é a imagem esquerda (vermelha) ou direita (azul/verde).

Todos os objetos do tipo `AnaglyphShaderAppearance` em um grafo de cena devem referenciar um único objeto `AnaglyphShaderAttributeSet` para que a construção do anaglifo seja feita de forma correta.

5.2.4 Classe BufferedImagePanel

A última classe do pacote `anaglyphstereoutils` estende a classe `JPanel` com a substituição de seu método `paint()` para a exibição de uma imagem contida em um objeto `BufferedImage`. Esta classe foi extraída de (Haseyama 2007).

5.2.5 Classe ContentBranch

Esta classe descreve o sub-grafo de conteúdo do grafo de cena Java 3D. Originalmente este código era parte da aplicação principal, porém com o crescimento do sub-grafo de conteúdo optou-se por criar uma classe separada para lidar separadamente com o mesmo. Esta prática permitiu um ganho de qualidade no software, que se tornou muito mais legível e organizado, diminuindo a ocorrência de erros no programa de um modo geral.

Esta é a classe que instancia o objeto `ShaderAttributeSet` que é utilizado por todos os objetos `ShaderAppearance` para passagem de parâmetros da aplicação para o shader programável.

5.2.6 CLASSE APPLETSTEREOVIEW

Esta é a classe de aplicação do sistema, onde está contido o método `main`. Os seus principais métodos são `init()`, que instancia e inicializa os objetos principais do sistema, e `run()`, que atualiza a imagem exibida em um `BufferedImagePanel`.

Esta classe implementa uma Thread que atualiza continuamente um objeto `BufferedImage`, exibido na tela por um container do tipo `BufferedImagePanel`. O objeto `BufferedImage` é a soma dos buffers de dois objetos `Canvas3D` no modo off-screen que contém as vistas esquerda e direita do observador. Este objeto é atualizado em resposta a uma chamada do método `createAnaglyph` da classe `AnaglyphstereoUniverse`.

Esta classe permite que o programa seja executado tanto como um aplicativo *stand-alone* quanto como um applet em uma janela de um navegador.

5.2.7 CLASSE ANAGLYPHSTEREOFRAME

Esta classe estende a classe `JFrame`, e implementa a interface gráfica do programa. Ela contém o objeto `BufferedImagePanel` que exibe o anaglifo na tela. Ela é instanciada pelo objeto `AppletStereoView`, que contém a função `main` do programa.

A interface implementada nesta classe consiste de dois menus, contendo as seguintes opções:

- sair do programa;
- alterar o modo de visualização do anaglifo;
- ajustar a separação interocular do observador;
- pausar/resumir a animação da cena.

As figuras abaixo ilustram esta configuração dos menus:

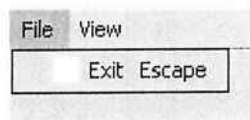

Figura 5-10: Classe AnaglyphStereoFrame: Menu File

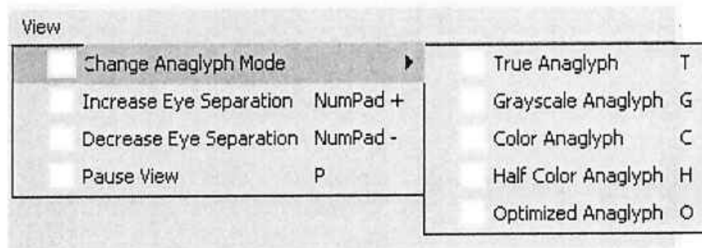

Figura 5-11: Classe AnaglyphStereoFrame: Menu View

Além da implementação dos menus, o outro aspecto importante desta classe é a sua construtora, onde o objeto `BufferedImage` que é atualizado pelo loop de execução do programa é associado ao objeto `BufferedImagePanel` que exibe a imagem contida em seu buffer.

5.2.8 SHADER PROGRAMÁVEL

O último aspecto importante do sistema é o shader programável utilizado para alterar a coloração das imagens na criação do anaglifo. O código-fonte deste shader, escrito em GLSL, é contido em um arquivo de texto comum que é lido pela classe `AnaglyphShaderAppearance` e gravado em um objeto do tipo `SourceCodeShader` para posterior associação a um objeto `ShaderProgram`.

A operação deste shader programável consiste em alterar a coloração final do pixel emitido de acordo com os parâmetros `leftImage` e `anaglyphMode`, determinados pela aplicação. O parâmetro `leftImage` indica se o pixel enviado pertence à imagem esquerda ou direita, e portanto qual tipo de coloração deve ser usado, vermelho ou azul/verde. O parâmetro `anaglyphMode` indica qual é o modo selecionado para produção do anaglifo, e portanto qual proporção de cada componente RGB do pixel recebido será utilizada para determinar a coloração final do pixel emitido. Para realizar esta operação, o shader utiliza duas estruturas de controle aninhadas: a estrutura externa é controlada pelo parâmetro `leftImage` e a estrutura interna é controlada pelo parâmetro `anaglyphMode`.

É importante notar que o programa não substitui a funcionalidade fixa de processamento de vértices do pipeline gráfico, apenas a funcionalidade de processamento de fragmentos. A substituição do processamento de vértices, caso fosse necessária, exigiria a utilização de outro shader programável do tipo *vertex shader*, que seria associado ao mesmo array de shaders (objeto `Shader[]`) que o

fragment shader, compondo assim um shader programável completo com as duas componentes; a de processamento de vértices e a de processamento de fragmentos.

6. RESULTADOS

Este capítulo analisa os resultados alcançados pelo sistema, avaliando se os objetivos estabelecidos inicialmente foram ou não atingidos.

6.1 AMOSTRA DO EFEITO ESTEREOSCÓPICO

A figura 6-1 é uma reprodução do efeito estereoscópico obtido pelo sistema em execução. A própria imagem impressa pode ser visualizada na forma estéreo com o uso de óculos anaglíficos, fornecendo uma amostra do efeito criado.

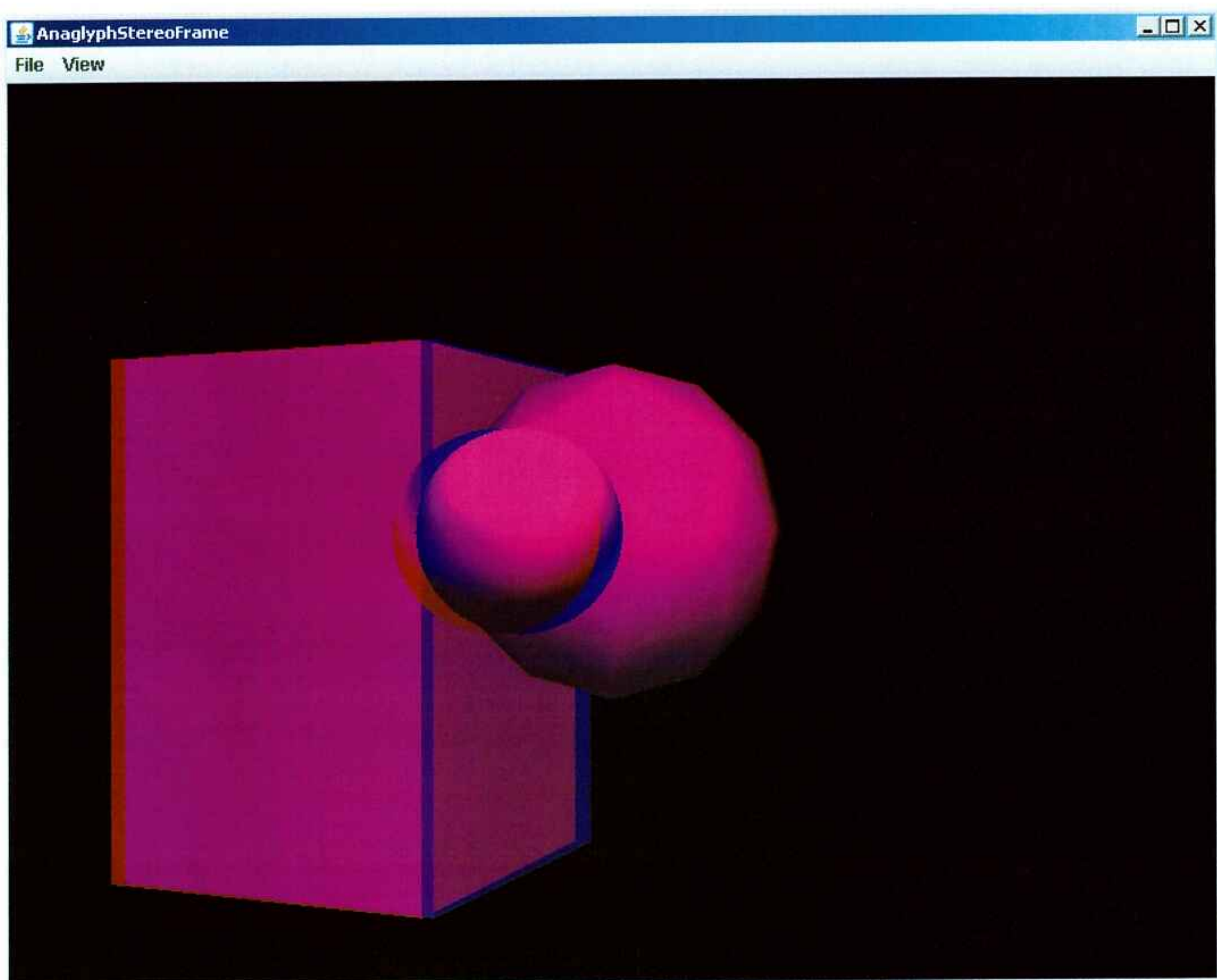

Figura 6-1: Amostra do efeito obtido

6.2 AVALIAÇÃO DO EFEITO ESTEREOSCÓPICO

A avaliação do efeito estereoscópico foi feita através da exibição do sistema em operação aos usuários de teste, por um período de 1 a 3 minutos. Dos usuários entrevistados:

- A maior parte dos usuários (60%) declarou ter percebido imediatamente a sensação de profundidade causada pelo efeito estéreo;
- 20% declararam perceber o efeito estereoscópico após um período de acomodação da visão;
- 20% não conseguiram perceber o efeito;
- Todos os usuários declararam ter percebido algum grau de persistência de cores na visão após retirar os óculos, como descrito no item 4.3;
- Nenhum usuário declarou ter sofrido reações adversas mais graves como dores de cabeça, náuseas, tontura ou mal-estar durante ou após a visualização do efeito no período avaliado;

6.3 DESEMPENHO DO SISTEMA EM TAXA DE QUADROS

Para obter uma medida do desempenho do sistema, foi necessário estender a classe `Canvas3D` e alterar o seu método `preRender()`, que é executado no início de cada quadro de renderização. Este método foi modificado com operações para gravar o número e a hora de início da renderização de cada quadro, obtendo uma estimativa da taxa de quadros média. A tabela a seguir mostra as medidas para os 50 primeiros quadros:

Tabela 6-1: Medidas dos atrasos na renderização dos quadros

Quadro nº:	Timestamp:	Atraso	Quadro nº:	Timestamp:	Atraso
0	1228070908296	0	25	1228070913551	12
1	1228070911647	3351	26	1228070913858	307
2	1228070911926	279	27	1228070913864	6
3	1228070911953	27	28	1228070913868	4
4	1228070911956	3	29	1228070913871	3
5	1228070911959	3	30	1228070913873	2
6	1228070911962	3	31	1228070913875	2
7	1228070911964	2	32	1228070913877	2
8	1228070911966	2	33	1228070914136	259
9	1228070912234	268	34	1228070914146	10
10	1228070912244	10	35	1228070914149	3
11	1228070912248	4	36	1228070914151	2
12	1228070912558	310	37	1228070914153	2
13	1228070912569	11	38	1228070914156	3
14	1228070912572	3	39	1228070914464	308
15	1228070912902	330	40	1228070914474	10
16	1228070912907	5	41	1228070914477	3
17	1228070912910	3	42	1228070914481	4
18	1228070912912	2	43	1228070914483	2
19	1228070912915	3	44	1228070914793	310
20	1228070913198	283	45	1228070914804	11
21	1228070913212	14	46	1228070914807	3
22	1228070913214	2	47	1228070915076	269
23	1228070913217	3	48	1228070915085	9
24	1228070913539	322	49	1228070915088	3

Nota-se um atraso de 3351 ms na renderização do primeiro quadro, devido ao tempo de inicialização do sistema. Além disto, podemos perceber a existência de picos periódicos nos valores dos atrasos. O gráfico a seguir ilustra esta situação:

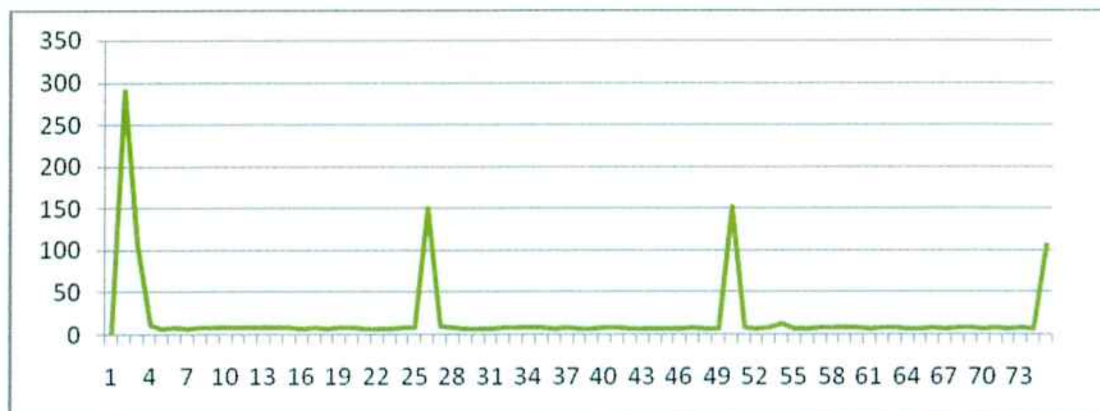

Figura 6-2: Picos de processamento do programa

Para maior clareza, apenas os 75 primeiros quadros de uma execução foram incluídos. Os atrasos máximos ocorrem a cada 25 quadros em média, indicando a ocorrência de picos de processamento nestes pontos. As possíveis causas desse comportamento são discutidas no item 7.2.

A taxa de quadros média foi determinada sobre uma amostra de 1800 quadros durante um período de 150 segundos, descartando-se os 50 primeiros quadros. O atraso médio exibido por esta amostra foi de 81 ms, resultando em uma taxa de quadros média de 12,35 quadros por segundo. As características do computador utilizado para a obtenção da amostra são as seguintes:

- Processador AMD Athlon dual core 64 X2 4600+, 2,41 GHz;
- 2 GB de memória RAM;
- Placa de vídeo *on-board* ATI Radeon X1200 series com 128 MB de memória compartilhada;
- Sistema operacional Windows XP Professional 2002, SP 2;
- Java VM versão 1.6.0;
- Java 3D versão 1.5.1.

7. CONCLUSÃO E TRABALHOS FUTUROS

7.1 CUMPRIMENTO DAS METAS PROPOSTAS

7.1.1 CRIAÇÃO DO EFEITO ESTEREOSCÓPICO

O objetivo principal do projeto consistia em obter a renderização estereoscópica de uma cena comum do Java 3D. Esta meta foi definitivamente alcançada, utilizando anaglifos para criar o efeito estéreo. Além de renderizar a cena em anaglifo puro, o sistema possui ainda outros 4 modos de renderização do anaglifo, bem como a capacidade de ajustar a separação interocular do observador para obter o melhor efeito.

7.1.2 ANIMAÇÃO

Outro requisito importante do programa é a capacidade de utilizar animações nos grafos de cena renderizados. Esta possibilidade foi mantida, com a atualização do anaglifo ocorrendo de forma dinâmica. A única limitação neste quesito é a taxa de quadros alcançada pelo sistema, que pode prejudicar efeitos de animação que requeiram uma continuidade maior.

7.1.3 INTERATIVIDADE

A interatividade é outro requisito principal das aplicações a que este projeto se destina, e foi mantida parcialmente na versão final do programa. O problema encontrado neste aspecto foi a captura dos eventos de entrada pelo programa, que necessita de outra janela da aplicação além da janela de exibição do anaglifo. O item 7.2 propõe uma melhoria que elimina este problema.

7.1.4 DESEMPENHO

A utilização de *fragment shaders* para obter a cor de cada pixel melhorou efetivamente o desempenho do sistema em comparação com a implementação existente no início deste projeto. A taxa de quadros reportada na primeira implementação era de 4,5 quadros por segundo. O programa desenvolvido neste

projeto obteve uma taxa de quadros de 12,35 quadros por segundo, representando uma melhoria expressiva em relação ao anterior.

7.2 Limitações do Sistema e Possíveis Melhorias

Aqui discutimos as principais limitações do sistema nos quesitos desempenho e funcionalidade, propondo algumas soluções possíveis para contornar estas limitações.

7.2.1 Desempenho

Os principais fatores que influenciaram negativamente o desempenho do sistema foram, em ordem decrescente de importância:

- A soma das imagens que compõe o anaglifo;
- A renderização adicional de dois `Canvas3D` no modo *offscreen*;

A renderização de um grafo de cena do Java 3D tem um custo computacional elevado; naturalmente, a geração de duas renderizações extras de uma cena causará um certo impacto na execução.

O outro fator que causa um atraso elevado ao gerar o anaglifo é o algoritmo de soma das duas imagens. Este algoritmo consiste de uma iteração que executa um número de vezes igual ao número de pixels da imagem renderizada. Para uma resolução de 800 x 600 pixels, isto gera um total de 480.000 iterações. Este algoritmo é executado inteiramente pela CPU, não utilizando nenhum recurso de aceleração gráfica.

Para se ter uma idéia do impacto destes dois itens sobre o desempenho geral do programa, utilizou-se a ferramenta de análise de perfil (profiler) do ambiente de desenvolvimento Netbeans IDE. Foram obtidos os seguintes resultados:

- A operação de soma das imagens utilizou 78,7% do tempo de processamento total do programa.
- As duas operações de renderização *offscreen* utilizam 15,4% do tempo de processamento total.

Somadas, estas duas operações são responsáveis por 94,1% do processamento total realizado pelo programa.

Uma forma de melhorar o desempenho destas duas funcionalidades seria a combinação das duas operações, realizando o segundo passo de renderização junto com a soma das imagens. Para isto, os dois canvas off-screen devem ser capazes de utilizar o mesmo buffer de renderização. O primeiro passo é realizado normalmente; ao realizar o segundo passo, cada pixel calculado é somado diretamente ao resultado da primeira renderização. Desta forma, ao término do processo o buffer irá conter a imagem com anaglifo pronta para ser exibida na tela.

7.2.2 FUNCIONALIDADE

A interatividade do sistema foi limitada pelo fato de se utilizar um `BufferedImagePanel` para exibir o anaglifo, dado que não é possível manipular o buffer de um canvas on-screen. Desta forma, a captura dos eventos de entrada ficou prejudicada, pois exige a utilização de uma janela à parte com uma área contendo o Canvas on-screen aonde é feita a captura destes eventos.

Esta limitação pode ser solucionada acrescentando rotinas de tratamento de eventos de entrada à classe `BufferedImagePanel`. Os eventos capturados podem ser enviados diretamente para o Canvas *on-screen*, que realiza a atualização dos objetos da cena.

Outro detalhe é que o uso de mapeamento de texturas do Java 3D em conjunto com o shader anaglífico não foi testado. Como o mapeamento de texturas exige uma operação de amostragem normalmente realizada pela funcionalidade fixa do pipeline gráfico, é provável que o acréscimo de alguma operação ao shader anaglífico seja necessário para a utilização conjunta com o mapeamento de texturas.

Uma solução capaz de eliminar todas estas limitações de uma só vez, passando a maior parte do esforço computacional necessário para o hardware de aceleração gráfica, é o uso de *Geometry Shaders*. Este tipo de shaders foi introduzido na versão 4.0 do Shader Model, porém o Java 3D ainda não oferece suporte a ele. O suporte no OpenGL é possível através de uma extensão.

Um *geometry shader* é capaz de gerar novos vértices a partir das primitivas recebidas; isto permite realizar todas as operações para a produção do anaglifo em apenas um passo de renderização. Cada vértice da imagem original originaria dois vértices separados por uma distância correspondente à paralaxe entre os olhos do

observador, e então um *fragment shader* seria utilizado para determinar a cor correspondente dos pixels para gerar o anaglifo.

7.3 APLICABILIDADE EM AMBIENTES DE REALIDADE AUMENTADA ESPACIAL

Conforme foi enunciado no início deste documento, o objetivo principal do projeto era desenvolver um sistema de visão estereoscópica para aplicação em ambientes com RAE, ou seja, a forma de uso final do sistema será através de projetores e não em uma tela de monitor comum. Embora a saída gerada pelo programa não deva se alterar conforme o modo de projeção, pequenas alterações ainda serão necessárias para otimizar o uso do sistema em ambientes de RAE, a saber, ajustes devem ser feitos para que o sistema possa funcionar em modo de tela cheia, projetando no ambiente apenas os objetos de interesse para a aplicação, isto é, ocultando botões de interface, barras de menu, e etc.

Além disso, novos testes devem ser feitos para ajustar os parâmetros que definem a paralaxe entre as duas imagens do par estereoscópico, para que o resultado obtido seja adequado à forma de projeção. Estes parâmetros são, principalmente, a distância entre as duas câmeras (ou olhos) e o seu ponto focal, que define uma distância mínima a partir da qual a paralaxe passa a ser negativa, isto é, objetos mais próximos aparentam 'saltar' para fora da tela.

7.4 INTEGRAÇÃO COM O ENJINE

O EnJine é um projeto do Laboratório de Tecnologias Interativas (INTERLAB) da Escola Politécnica da Universidade de São Paulo, que estabelece um framework para o desenvolvimento de jogos eletrônicos utilizando o Java 3D. Ele também pode ser utilizado em jogos e aplicações de Realidade Aumentada.

Uma das motivações para este projeto foi a possibilidade de incorporar um módulo de estereoscopia ao enJine. A integração com o enJine era um objetivo adicional do projeto que infelizmente não pôde ser cumprido. Mesmo não tendo sido integrado ao enJine, este projeto deu origem a um framework de estereoscopia dentro do Java 3D que deve ser facilmente integrável com outros aplicativos e frameworks, incluindo aí o projeto do INTERLAB.

A integração deste projeto com o framework do INTERLAB possibilitará o suporte nativo a aplicações estereoscópicas neste, somando-se à já variada gama de funcionalidades oferecidas por este.

Para incluir o suporte à estereoscopia com anaglifos no enJine, será necessário modificar suas classes que contém o sub-grafo de visualização da cena para que estas suportem a utilização de dois objetos `Canvas3D` no modo *off-screen*, renderizando ambas as imagens de um par estereoscópico. Além disso um método para soma destas imagens deve ser introduzido, possivelmente na mesma classe que renderiza os canvas off-screen.

8. BIBLIOGRAFIA

3dtv.at. *3dtv.at - Anaglyph Methods Comparison.* 2005. http://3dtv.at/Knowhow/AnaglyphComparison_en.aspx (acesso em 18 de 09 de 2008).

ALBUQUERQUE, ANTONIA LUCINELMA PESSOA. *UM MODELO PARA VISUALIZAÇÃO ESTEREOSCÓPICA UTILIZANDO WEBCAMS.* Rio de Janeiro: PUC Rio - Pontifícia Universidade Católica do Rio de Janeiro, 2006.

ANDRADE, L. "Seminário para disciplina SCE 5799 - Computação Gráfica." 2007.

Azuma, Ronald T. *SIGGRAPH '97 Course Notes #30: Making Direct Manipulation Work in Virtual Reality.* Malibu, CA: Hughes Research Laboratories, 1997.

Azuma, Ronald T. *A Survey of Augmented Reality.* Artigo Científico, Malibu, CA: Hughes Research Laboratories, 1997.

—. *The Challenge of Making Augmented Reality Work Outdoors.* Malibu, CA: HRL Laboratories, 1999.

Azuma, Ronald T., et al. *Making Augmented Reality Work Outdoors Requires Hybrid Tracking.* San Francisco, CA: Proceedings of the First International Workshop on Augmented Reality, 1998.

Azuma, Ronald, Bruce Hoff, Howard Neely, e Ron Sarfaty. *A Motion-Stabilized Outdoor Augmented Reality System.* Houston, TX: Proceedings of IEEE Virtual Reality '99, 1999.

Azuma, Ronald, e Gary Bishop. *Improving Static and Dynamic Registration in an Optical See-Through HMD.* Chapel Hill, NC: University of North Carolina at Chapel Hill, 1996.

Azuma, Ronald, Howard Neely, Mike Daily, e Jon Leonard. *Performance Analysis of an Outdoor Augmented Reality Tracking System that Relies Upon a Few Mobile Beacons.* Santa Barbara, CA: ISMAR, 2006.

Broll, Wolfgang, et al. *ARTHUR: A Collaborative Augmented Environment for Architectural Design and Urban Planning.* Sankt Augustin, Germany: Fraunhofer Institute for Applied Information Technology (FIT), 2004.

Fernandes, António Ramires. *Tópicos Avançados de Computação Gráfica - GLSL - Programação de Shaders.* 2006.

Geiger, Christian, Leif Oppermann, e Christian Reimann. *3D-Registered Interaction-Surfaces in Augmented Reality Space.* Halberstadt: Hochschule Harz, Paderborn University C-LAB, 2003.

General Constraints for Batch Multiple-Target Tracking Applied to Large-Scale Videomicroscopy. Lausanne, Switzerland: Ecole Polytechnique Federale de Lausanne, 2008.

Haseyama, Adriana. *Projeções Estereoscópicas para Realidade Aumentada.* São Paulo: POLI - USP, 2007.

Hoff, Bruce, e Ronald Azuma. *Autocalibration of an Electronic Compass in an Outdoor Augmented Reality System.* Munique, Alemanha: International Symposium on Augmented Reality 2000, 2000.

Hoff, William A., Khoi Nguyen, e Torsten Lyon. *Computer vision-based registration techniques for augmented reality.* Boston, MA: Colorado School of Mines, Division of Engineering, 1996.

Holloway, Richard Lee. *Registration Errors in Augmented Reality Systems.* Chapel Hill: University of North Carolina at Chapel Hill, 1995.

John Kessenich, Dave Baldwin, Randi Rost. *The OpenGL Shading Language.* 3dLabs, 2006.

Kirner, Cláudio, e Romero Tori. *Introdução à Realidade Virtual, Realidade Misturada e.* São Paulo, 2004.

Lepetit, Vincent. *On Computer Vision for Augmented Reality.* Lausanne, Suíça: École Polytechnique F´ed´erale de Lausanne (EPFL), 2008.

Lepetit, Vincent, e Pascal Fua. *Monocular Model-Based 3D Tracking of Rigid Objects: A Survey.* now, 2005.

Lepetit, Vincent, Luca Vacchetti, Daniel Thalmann, e Pascal Fua. *Fully Automated and Stable Registration for Augmented Reality Applications.* Lausanne, Switzerland: Swiss Federal Institute of Technology, 2004.

Manssour, Isabel Harb. *Introdução a Java 3D.* Porto Alegre: Faculdade de Informática - PUCRS, 2003.

McGraw-Hill. *McGraw-Hill Encyclopedia of Science and Technology.* McGraw-Hill, 2004.

Milgram, Paul, Haruo Takemura, Utsumi Akira, e Fumio Kishino. *Augmented Reality: A class of displays on the reality-virtuality continuum.* Telemanipulator and Telepresence Technologies, SPIE V.2351., 1994.

Mundo e Vestibular. *O Olho e a Visão.* 2007. http://www.mundovestibular.com.br/articles/469/1/O-OLHO-E-A-VISAO/Paacutegina1.html (acesso em 18 de 11 de 2008).

Ohta, Yuichi, e Hideyuki Tamura. *Mixed Reality: Merging Real and Virtual Worlds.* Springer-Verlag, 1999.

Raposo, Alberto B., Flávio Szenberg, Marcelo Gattass, e Waldemar Celes. *Visão Estereoscópica, Realidade Virtual, Realidade Aumentada e Colaboração* . Rio de Janeiro: Departamento de Informática, PUC-Rio , 2004.

Raskar, R., Welch, G., Fuchs, H. *Spatially Augmented Reality.* San Francisco: First International Workshop on Augmented Reality, 1998.

—. *Spatially Augmented Reality.* San Francisco: First International Workshop on Augmented Reality, 1998.

Redmond, Kent C., e Thomas M. Smith. *Project Whirlwind: The History of a Pioneer Computer.* Bedford, MA: Digital Press, 1980.

Río, A. del, J. Fischer, M. Köbele D. Bartz, e W. Straßer. *Augmented Reality Interaction for Semiautomatic Volume Classification.* Tübingen, Germany: University of Tübingen, Germany, 2005.

Rushforth, Kevin, e Chien Yang. "Tha Java 3D API." *Apresentação realizada na JavaOneSM Conference, 2005.* Sun Microsystems, 2005.

Schmalstieg, Dieter, et al. *The Studierstube Augmented Reality Project.* Vienna, Austria: Vienna University of Technology, 2002.

Segal, Mark, e Kurt Akeley. *The Design of the OpenGL Graphics Interface.* Mountain View: Silicon Graphics Computer Systems, 1994.

—. *The OpenGL Graphics System: A Specification.* Silicon Graphics, 2006.

Sevo, Daniel. *Becoming a Computer Animator.* 01 de 01 de 2005. http://www.danielsevo.com/ (acesso em 14 de 08 de 2008).

Slay, Hannah, Bruce Thomas, e Rudi Vernik. *Tangible User Interaction Using Augmented Reality.* Mawson Lakes, South Australia: University of South Australia, 2001.

Sousa, A. Augusto de. *Fisiologia e Percepção em Ambientes* . FEUP/DEEC, 2006.

State, Andrei, Gentaro Hirota, David T. Chen, William F. Garrett, e Mark A. Livingston. *Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking.* Chapel Hill: Department of Computer Science University of North Carolina at Chapel Hill, 1997.

StereoGraphics Corporation. *Stereo Graphics Developer's Handbook.* StereoGraphics Corporation, 1997.

Sun Microsystems. *Java 3D Concepts.* 2001. http://download.java.net/media/java3d/javadoc/1.4.0/javax/media/j3d/doc-files/Concepts.html (acesso em 10 de 09 de 2008).

—. *The Java 3D API Specification.* Palo Alto: Sun Microsystems, 2000.

—. *The Java 3D Documentation Pages.* 2003. http://java.sun.com/javase/technologies/desktop/java3d/forDevelopers/J3D_1_3_API/j3dapi/javax/media/j3d/package-summary.html (acesso em 19 de 09 de 2008).

Sutherland, Ivan. *A HEAD-MOUNTED THREE-DIMENSIONAL DISPLAY.* 1969.

TOMMASELLI, A. M. G. *Fotogrametria Básica - Estereoscopia e Paralaxe.* Presidente Prudente, 2006.

Tori, Romero, Claudio Kirner, e Robson Siscoutto. *Fundamentos e Tecnologia de Realidade Virtual e Aumentada.* Belém: SBC – Sociedade Brasileira de Computação, 2006.

Veigl, Stephan, Andreas Kaltenbach, Florian Ledermann, Gerhard Reitmayr, e Dieter Schmalstieg. *Two-Handed Direct Interaction with ARToolKit.* Viena, Áustria: Vienna University of Technology, Austria, 2002.

Ward, Mark, Ronald Azuma, Robert Bennett, Stefan Gottschalk, e Henry Fuchs. *A Demonstrated Optical Tracker With Scalable Work Area for Head-Mounted Display Systems.* Chapel Hill, NC: University of North Carolina, 1999.

Wellner, P., W. Mackay, e R. Gold. *Special issue on computer augmented environments: back to the real world.* 1993.

Wells, Don. *Extreme Programming.org.* 1999. http://www.extremeprogramming.org (acesso em 10 de 10 de 2008).

You, Suya, Ulrich Neumann, e Ronald Azuma. *Hybrid Inertial and Vision Tracking for Augmented Reality Registration.* Los Angeles, CA: University of Southern California, 1999.

—. *Orientation Tracking for Outdoor Augmented Reality Registration.* Plzen, Czech Republic: Computer Science Dept., University of West Bohemia, 1999.

Zalman Cool Innovations. *Zalman Cool Innovations.* 2007. <http://www.zalman.co.kr/eng/product/view.asp?idx=331&code=032> (acesso em 18 de 11 de 2008).