ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
ENGENHARIA MECATRÔNICA

CHRISTIANO ODIR CARDOSO MEIRELLES

Ferramenta de visualização tridimensional de tubos flexíveis e cabos umbilicais

São Paulo
2010

CHRISTIANO ODIR CARDOSO MEIRELLES

Ferramenta de visualização tridimensional de tubos flexíveis e cabos umbilicais

Trabalho de Formatura apresentado à Escola Politécnica da Universidade de São Paulo para obtenção de graduação em Engenharia Mecatrônica.

Área de Concentração: Engenharia Mecatrônica.

Orientador: Clóvis de Arruda Martins

São Paulo
2010

FICHA CATALOGRÁFICA

Meirelles, Christiano Odir Cardoso

Ferramenta de visualização tridimensional de tubos flexíveis e cabos umbilicais / C.O.C. Meirelles. -- São Paulo, 2010.

119 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Tubos flexíveis 2. Cabos umbilicais 3. Petróleo (Exploração) 4. CAD I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecâ-nicos II. t.

Agradecimentos

À Rodrigo Provasi Correa por auxiliar na orientação e desenvolvimento deste projeto.

Ao Professor Doutor Ricardo Nakamura pelo auxílio no estudo teórico de computação gráfica.

Aos integrantes do Núcleo de Dinâmica e Fluidos (NDF-EPUSP) pelo apoio e incentivo no decorrer de todo o projeto.

Resumo

Os cabos umbilicais e tubos flexíveis são elementos utilizados na exploração offshore de petróleo, sendo que o primeiro tem a finalidade de interligar uma unidade flutuante a um poço submerso, executando diversas funções de controle, enquanto que o segundo é destinado ao transporte de fluidos como petróleo bruto, óleo, água ou gás. A especificação e o projeto destes tipos de cabos é uma tarefa bastante complexa devido à grande variedade de componentes e arranjos possíveis, sendo dependentes da funcionalidade que é desejada para o cabo, além de características estruturais como o peso e a resistência axial. Com base neste cenário, foi desenvolvida uma ferramenta CAD para projeto e visualização bidimensional de tubos flexíveis e cabos umbilicais. Muitas vezes, no entanto, apenas uma visualização da secção transversal do mesmo não é suficiente, uma vez que temos elementos dispostos em helicóides e um grande número de componentes presentes. Este projeto tem por finalidade a construção de um módulo de visualização tridimensional do cabo projetado pelo software descrito anteriormente. Este novo módulo pode auxiliar sobremaneira o projeto, pois permite uma visualização mais realista do cabo sendo desenvolvido, possibilitando muitas vezes encontrar inconsistências ainda em fase de projeto. O módulo tridimensional tem por objetivo a representação mais próxima possível da real, e desta forma, são levados em conta efeitos de iluminação e textura, além da implementação de realidade virtual (estereoscopia).

Palavras chave: *Tubos Flexíveis, Cabos Umbilicais, Petróleo (Exploração), CAD.*

Abstract

Umbilical cables and flexible pipes are elements employed in offshore oil exploitation. The purpose of the first is to connect the floating platform to the submerged well, performing control functions, while the second is intended for the transport of fluids such as crude oil, water or gas. The specification and design of these types of cables is a very complex task due to the wide variety of components and possible arrangements, being dependent on the functionality that is desired for the cable, in addition to structural features and the weight and shear strength. Based on this scenario, a CAD tool for design and visualization of two-dimensional flexible pipes and umbilical cables was developed. Often, however, only a cross-sectional view is not enough, since there are elements arranged in helices and a large number of components present. This project aims to build a module for three-dimensional visualization of the cable designed by the software described above. This new module can help the project as it enables a more realistic view of the cable being developed, allowing to find inconsistencies in the design phase. The three-dimensional module being developed aims the closest possible representation of reality and thus will takes into account texture and lighting effects and the implementation of virtual reality (stereoscopy).

Key words: *Flexible pipes, Umbilical cables, Oil (exploitation), CAD.*

Índice de Figuras

Figura 1.1 - Exemplo de cabos umbilicais – Extraído de (1) 10
Figura 1.2 - Esquema do projeto de um cabo umbilical 12
Figura 1.3 - Esquema de tubo flexível – Extraído de (2) 12
Figura 2.1 - Janela principal do módulo bidimensional 15
Figura 2.2 - Estrutura de armazenamento de dados do módulo bidimensional 16
Figura 3.1 - Ângulos para a parametrização da esfera – Extraído de (3) 21
Figura 3.2 - Exemplo de malha de um cilindro utilizando elementos triangulares 22
Figura 3.3 - Operações booleanas com sólidos – Extraído de (4) 23
Figura 3.4 - Exemplos de sólidos gerados por revolução – Extraído de (5) 24
Figura 3.5 - Exemplo de sólido gerado por extrusão 26
Figura 3.6 - Caminho percorrido (esquerda) e sólido gerado por arrasto (direita) 27
Figura 3.7 - Exemplo do ângulo entre normal e tangente da curva constantes 27
Figura 3.8 - Exemplo de rotação de um perfil centrado no centro de curvatura da curva 28
Figura 3.9 - Exemplo de rotação do perfil em seu próprio centro e posterior translação 29
Figura 4.1 - Exemplo de transformação de translação 32
Figura 4.2 - Ângulos utilizado na rotação 32
Figura 4.3 - Exemplo de rotação em torno de um eixo qualquer 34
Figura 4.4 - Exemplo de transformação de escala 35
Figura 4.5 - Exemplo de transformação de cisalhamento 36
Figura 4.6 - Exemplo de composição de transformações 39
Figura 4.7 - Elementos básicos da projeção 40
Figura 4.8 - Exemplo de projeção Perspectiva (a) e Paralela (b) – Extraído de (6) 41
Figura 4.9 - Parâmetro da projeção paralela 43
Figura 4.10 - Parâmetros da projeção perspectiva 45
Figura 5.1 - Espaço de cores RGB – Extraído de (7) 48
Figura 5.2 - Espaço de cores CMY – Extraído de (8) 49
Figura 5.3 - Espaço de cores HSV – Extraído de (7) 50
Figura 5.4 - Espaço de cores HSL – Extraído de (7) 51
Figura 5.5 - Fonte luminosa direcional – Extraído de (3) 53
Figura 5.6 - Fonte luminosa posicional – Extraído de (3) 53

Figura 5.7 - Fonte luminosa tipo 'Spot Light' – Extraído de (3) 54
Figura 5.8 - Exemplo de iluminação - Componente ambiente 55
Figura 5.9 - Exemplo de iluminação - Componente difusa 56
Figura 5.10 - Exemplo de iluminação - Componente especular 58
Figura 5.11 - Exemplo de imagem gerada por Ray-Tracing – Extraído de (9) 59
Figura 5.12 - Exemplo de aplicação de Flat Shading 61
Figura 5.13 - Exemplo de aplicação de shadder – Algoritimo de Gourraud 62
Figura 5.14 - Coordenadas no mapeamento de texturas – Extraído de (3) 63
Figura 5.15 - Cubo mapeado com textura – Adaptado de (10) 64
Figura 5.16 - Exemplo de terreno gerado através de mapeamento de deslocamentos – Adaptado de (11) 65
Figura 5.17 - Exemplo de aplicação de 'Bump Map' – Extraído de (12) 66
Figura 5.18 - Exemplo de normal map – Extraído de (13) 66
Figura 6.1 - Visualização por um ponto de observação único – Extraído de (14) 68
Figura 6.2 - Campo de visão dos olhos esquerdo e direito – Extraído de (14) 68
Figura 6.3 - Representação da distância inter-axial – Extraído de (14) 69
Figura 6.4 - Representação da profundidade da tela – Extraído de (14) 69
Figura 6.5 - Exemplo de paralaxe – Extraído de (14) 70
Figura 6.6 - Variação da paralaxe – Extraído de (14) 70
Figura 6.7 - Exemplo de estereoscopia por cores – Extraído de (15) 72
Figura 6.8 - Funcionamento da estereoscopia por frequência – Extraído de (14) 73
Figura 7.1 - Estereoscopia por cores utilizando WPF – Extraído de (3) 80
Figura 8.1 - Esboço inicial da interface do módulo tridimensional 85
Figura 9.1 - Modelo em Espiral 87
Figura 9.2 - Diagrama de Fluxo de Dados Nível 0 87
Figura 9.3 - Diagrama de Fluxo de Dados Nível 1 88
Figura 9.4 - Diagrama de Fluxo de Dados Nível 2 (Unidade de Processamento Gráfico) .. 89
Figura 9.5 - Diagrama de Fluxo de Dados Nível 2 (Interface Gráfica) 90
Figura 9.6 - Diagrama de Casos de Uso 91
Figura 10.1 - Diagrama de Dependências de Bibliotecas 99
Figura 10.2 - Exemplo de capa plástica gerada por revolução 101
Figura 10.3 - Exemplo de armadura helicoidal gerada por arrasto 101
Figura 10.4 - Exemplo de mangueiras geradas por arrasto 102
Figura 10.5 - Exemplo de perfil da carcaça inter-travada 102

Figura 10.6 - Exemplo de carcaça inter-travada 103
Figura 10.7 - Exemplo de cabo eletro-hidráulico com elementos básicos 104
Figura 10.8 - Modelo básico de funcionamento da 'Trackball' – Extraído de (19) 105
Figura 10.9 - Exemplo de funcionamento da 'Trackball' – Extraído de (19)..... 106
Figura 10.10 - Sistema de coordenadas do elemento gráfico – Extraído de (19)..... 106
Figura 10.11 - Sistema de coordenadas da esfera – Extraído de (19)..... 107
Figura 10.12 - Exemplo para cálculo do eixo e ângulo de rotação..... 108
Figura 10.13 - Parâmetros da câmera perspectiva para cálculo de xMax..... 109
Figura 10.14 - Parâmetros da câmera ortográfica para cálculo de xMax 110
Figura 10.15 - Diálogo de Configuração de Iluminação 111
Figura 10.16 - ComboBox de seleção de texturas 112
Figura 10.17 - Exemplo de tubo flexível com utilização de texturas 113
Figura 10.18 - Cabo eletro hidráulico visto em modo de realidade virtual 114

Sumário

Capítulo 1. Introdução 10

Capítulo 2. Módulo de Projeto e Visualização Bidimensional 15

Capítulo 3. Geração de Sólidos Tridimensionais 19

3.1. Geração de sólidos por parametrização 19

3.2. Geração de sólidos por operações booleanas 22

3.3. Geração de sólidos por revolução 23

3.4. Geração de sólidos por extrusão e arrasto 25

Capítulo 4. Transformações e Projeções 31

4.1. Transformações Básicas 31

4.2. Coordenadas Homogêneas 36

4.3. Composição de Transformações 38

4.4. Projeções 39

4.4.1. Matriz de Visualização 41

4.4.2. Matriz de Projeção 42

Capítulo 5. Iluminação e Texturas 47

5.1. Técnicas de Iluminação 51

5.1.1. Principais Tipos de Fontes Luminosas 52

5.1.2. Reflexão Local 54

5.1.3. Reflexão Global 58

5.1.4. Shaders 60

5.2. Texturas 62

Capítulo 6. Realidade Virtual 67

6.1. Por Cores 71

6.2. Por Frequência 72

6.3. Por Polarização 73

Capítulo 7. Requisitos e Bibliotecas Gráficas 74

7.1. Requisitos Gráficos 74

7.2. OpenGL 76

7.3. WPF 3D 78

7.4. XNA *80*
7.5. Comparação e Escolha da Biblioteca Gráfica *81*
Capítulo 8. Especificações do Módulo Tridimensional **84**
Capítulo 9. Projeto do Software **86**
9.1. Diagramas de Fluxo de Dados *87*
9.2. Casos de Uso *91*
9.2.1. Descrição dos Casos de Uso 92
Capítulo 10. Implementação do Módulo Tridimensional **99**
10.1. Estrutura do Software *99*
10.2. Geração tridimensional dos componentes *100*
10.3. Ferramentas de Visualização *104*
10.3.1. Rotação 105
10.3.2. Translação 108
10.3.3. Escala 110
10.4. Iluminação *111*
10.5. Texturas *112*
10.6. Realidade Virtual *113*
Capítulo 11. Conclusão **115**
Capítulo 12. Bibliografia **117**

Capítulo 1. Introdução

Dada a importância crescente do petróleo na economia e política global, assim como na produção de combustíveis, plásticos, lubrificantes, energia elétrica dentre outros, é necessária uma tecnologia cada vez mais aprimorada e eficiente, que busque maiores resultados com o menor custo possível.

No Brasil, a grande maioria das fontes de petróleo estão localizadas em alto mar. Em função da crescente demanda, a extração é aumentada, em poços de grandes profundidades, que ultrapassam 2000 m.

Para a extração do petróleo em poços submersos, são necessários cabos umbilicais, os quais proporcionam controle hidráulico das maquinas submersas, assim como suprimento de energia, controle dos equipamentos e em alguns casos, até mesmo, a injeção de fluidos químicos. A Figura 1.1 apresenta alguns exemplos de cabos umbilicais.

Figura 1.1 - Exemplo de cabos umbilicais – Extraído de (1)

O projeto de um cabo umbilical é bastante complexo e específico para cada aplicação. Sua primeira etapa é a definição das funções que devem ser executadas pelo mesmo, que resulta nos componentes a serem empregados, como cabos elétricos, mangueiras hidráulicas, condutores de fibra ótica, mangueiras de injeção, entre outros. A segunda etapa é a definição do arranjo relativo desses componentes na seção transversal. A terceira etapa consiste na definição dos elementos estruturais necessários, armaduras e camadas plásticas, para manter a integridade e as funções do cabo umbilical durante a instalação e a operação.

Definido o arranjo geométrico e estrutural do cabo umbilical, podem ser calculados os parâmetros de rigidez, permitindo que seja realizada a análise mecânica global, onde são calculados os esforços solicitantes em cada seção transversal. Posteriormente é realizada a análise mecânica local, na qual, a partir dos esforços solicitantes, é determinada a distribuição de tensões em cada um dos elementos que compõem a seção transversal.

Definido o projeto de um cabo umbilical, é construído um protótipo, que é submetido a uma série de testes e ensaios. Se os resultados desses testes não forem adequados, é necessário reprojetar o cabo, construir um novo protótipo e realizar uma nova bateria de ensaios, o que, certamente, implica em custos e prazos. A Figura 1.2 apresenta, de forma esquemática, as etapas do projeto de um cabo umbilical.

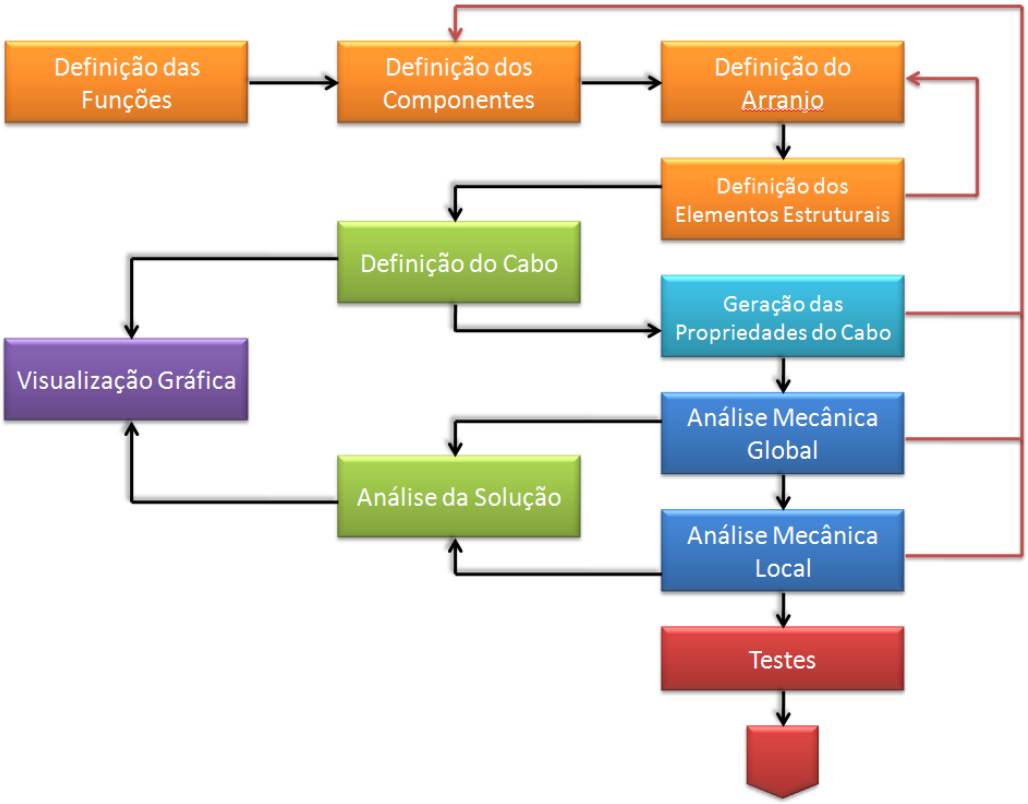

Figura 1.2 - Esquema do projeto de um cabo umbilical

Além dos cabos umbilicais, existem os tubos flexíveis, de construção semelhante, porém com função diversa: o transporte de fluídos como petróleo bruto, óleo, água ou gás do poço submerso à unidade flutuante ou, no sentido contrário, da unidade flutuante a uma tubulação submersa. A Figura 1.3 traz um esquema de um tubo flexível típico.

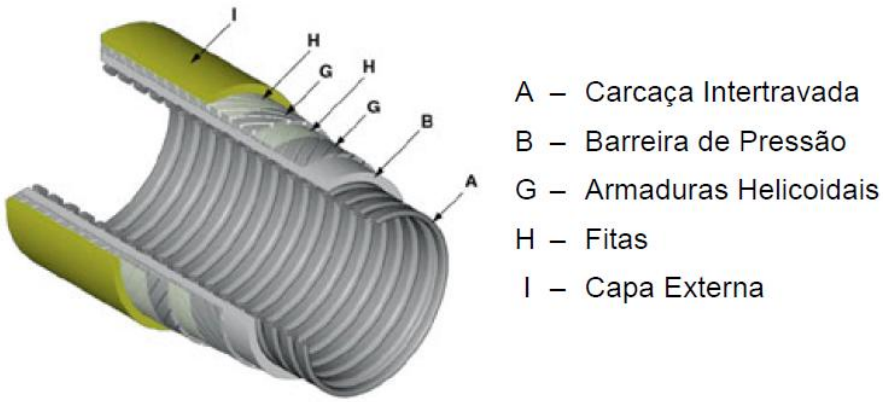

Figura 1.3 - Esquema de tubo flexível – Extraído de (2)

À medida que se aumenta a profundidade em que o petróleo é explorado no Brasil, aumenta a complexidade do projeto de um cabo umbilical ou tubo flexível. Concepções que

são tradicionalmente usadas podem se tornar inviáveis com o aumento da lâmina d'água. Recursos de computação gráfica podem auxiliar sobremaneira o projeto de um cabo umbilical ou tubo flexível, como uma ferramenta de CAD especificamente desenhada para essa função e ferramentas de visualização que permitam a construção de um protótipo virtual, antes da fabricação do protótipo real.

Com base neste cenário, foi desenvolvido um *software* CAD com o intuito de auxiliar no projeto destes tipos de cabos, permitindo uma visualização plana da secção transversal do mesmo, descrita em maiores detalhes no Capítulo 2.

Devido à grande complexidade destes tipos de cabos, muitas vezes apenas uma visualização plana não é suficiente para seu projeto, pois estão presentes uma grande variedade de componentes, além de diferentes disposições espaciais, como por exemplo, o formato helicoidal das armaduras de tração. Desta forma, este projeto tem o intuito de desenvolver um módulo adicional para a visualização tridimensional do cabo projetado.

Para seu desenvolvimento, foram estudadas algumas técnicas de computação gráfica, iniciando-se pela geração de modelos tridimensionais, descritas no Capítulo 3. Esta etapa é importante, pois é necessário um modelo que descreva completamente a geometria de cada um dos elementos, sua forma e posição no espaço, e como será feita sua malha tridimensional para que se obtenha o efeito desejado.

Posteriormente, estudou-se alguns fundamentos de computação gráfica como transformações, projeções, iluminação e texturas, descritas no Capítulo 4 e no Capítulo 5, sendo estes necessários para aplicação dos efeitos de renderização de forma correta.

O módulo tridimensional tem por objetivo, além da visualização do modelo, a utilização de recursos de iluminação e texturas. Outra funcionalidade incorporada ao *software*

é a implementação de realidade virtual (estereoscopia), apresentada no Capítulo 6, de modo a permitir uma visualização muito próxima da real ainda em fase de desenvolvimento do cabo.

Outro item importante a ser levado em conta no desenvolvimento de um programa de visualização tridimensional é a lingugem e a biblioteca gráfica utilizadas para implementação. O Capítulo 7 discorre sobre os requisitos levantados para o *software* desejado, bem como o estudo de três bibliotecas gráficas bastante conhecidas no marcado - o *OpenGL*, o *XNA*, e o *WPF 3D*-, comparando-as e definindo a mais compatível para utilização neste projeto.

Feito todo o estudo teórico e definida as ferramentas de trabalho, deu-se início ao projeto e desenvolvimento do software, descritos no Capítulo 9 e Capítulo 10.

Capítulo 2. Módulo de Projeto e Visualização Bidimensional

Devido à necessidade de uma ferramenta que permitisse o projeto e visualização de cabos umbilicais e tubos flexíveis, foi desenvolvido um software destinado a esta funcionalidade.

Este software permite a criação facilitada de tais cabos e sua respectiva visualização, pelo fato de possuir internamente os componentes comumente utilizados, de forma que o usuário deve somente selecionar o componente que deseja utilizar e inserir os parâmetros necessários. A janela principal deste software é mostrada na Figura 2.1.

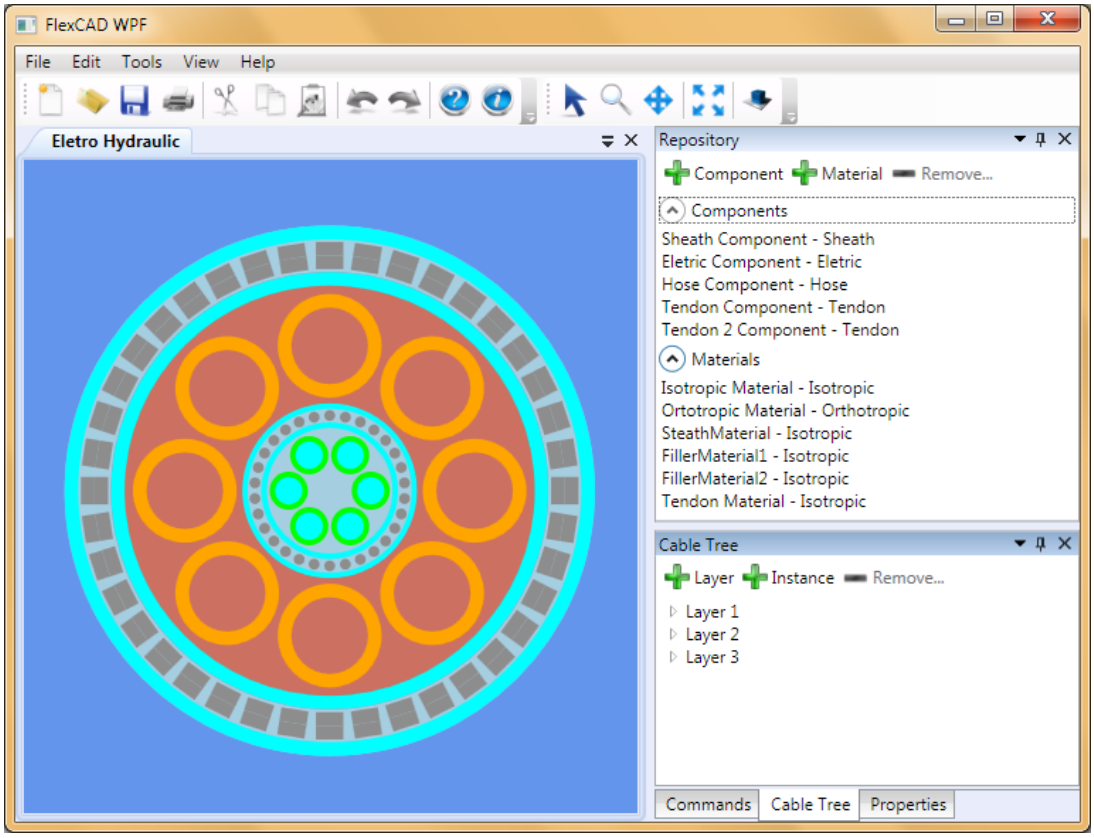

Figura 2.1 - Janela principal do módulo bidimensional

Nesta janela, é possível visualizar na região principal a seção transversal do cabo sendo projetado, e na direita, controles para seu desenvolvimento, como a lista de

componentes criados, as propriedades do objeto selecionado e uma visualização em árvore de sua estrutura.

Este software foi totalmente desenvolvido em C#, utilizando-se o NET Fremework 4.0 e para interface gráfica, adotou-se o WPF (Windows Presentation Foundation), um componente presente no .NET Framework. Devido a sua grande flexibilidade, como ferramenta de visualização bidimensional optou-se por manter o próprio WPF, evitando a necessidade de incluir novas bibliotecas gráficas ao projeto.

O software possui uma estrutura de armazenamento de dados também otimizada, que foi projetada para possibilitar o reuso de componentes e materiais, uma vez que é muito comum se ter em um cabo umbilical diversas mangueiras com as mesmas características, ou o mesmo material sendo utilizado em componentes diferentes, por exemplo. A Figura 2.1 mostra como foi definida esta estrutura.

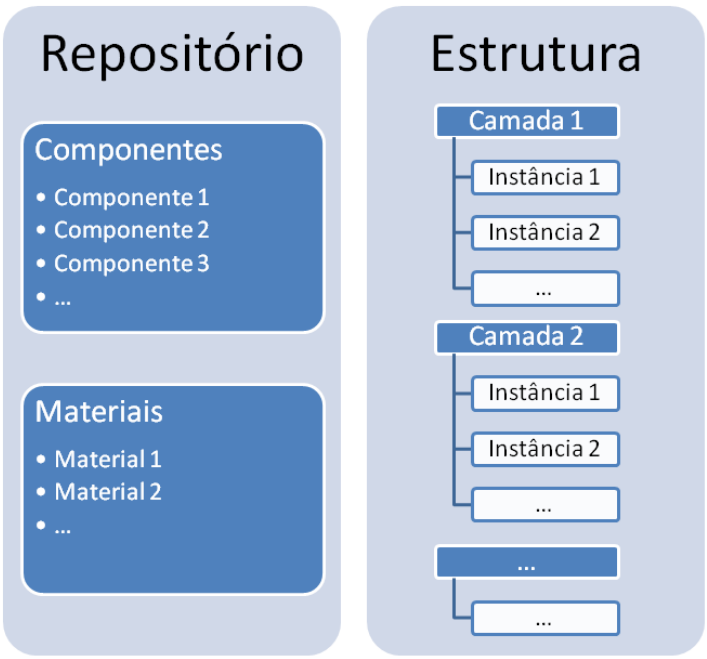

Figura 2.2 - Estrutura de armazenamento de dados do módulo bidimensional

Os dados são divididos em duas regiões distintas, uma correspondente aos componentes e materiais, funcionando como um repositório para o programa, e seus objetos

armazenados em forma de lista, enquanto a outra, em forma de árvore, é responsável por armazenar a estrutura do cabo propriamente dita, com as camadas e as instâncias dos componentes presentes no repositório.

Os componentes implementados internamente pelo software são:

- **Carcaça Intertravada (*Interlocked Carcass*):** Este elemento não está presente nos cabos umbilicais e corresponde à camada mais interna dos tubos flexíveis. Sua função é evitar o colapso do tubo e resistir às pressões externas ou de estrangulamento causadas pelas armaduras helicoidais.
- **Camada Plástica (*Jacket*):** A camada plástica está presentes tanto nos tubos flexíveis quanto nos cabos umbilicais, tendo a função de garantir o alinhamento dos tendões das armaduras helicoidais. No caso dos tubos flexíveis, possui ainda a funcionalidade de garantir a estanqueidade do tubo e o posicionamento dos componentes. Ainda, quando presentes do lado externo do cabo, sua finalidade passa a ser a proteção do mesmo de possíveis agressões do meio em que o tubo se encontra.
- **Armadura de Pressão (*Pressure Armor*):** Esse componente funciona como suporte aos demais, além de resistir a grandes pressões internas. A armadura de pressão pode ser encontrada com diversos tipos de perfis, portanto é necessário o desenho desse perfil, assim como sua espessura e raio médio.
- **Armaduras de Tração Helicoidais (*Armor*):** Sua principal função é fornecer resistência axial ao tubo. Essa camada aparece geralmente em pares enrolados de forma helicoidal em sentidos opostos. É composta por um determinado número de tendões com certo ângulo de inclinação os quais são, em sua maioria, feitos de aço.
- **Preenchimento (*Filler*):** Serve para o preenchimento interno em ocasiões em que ocorra um vazio entre os elementos, de forma a mantê-los devidamente alinhados. Têm-se

como exemplo bastante comum uma camada com mangueiras: o espaço entre elas poderia causar um desarranjo estrutural e consequentemente a falha do equipamento.

- **Núcleo Elétrico (*Eletric Cable*):** A energia é fornecida para os equipamentos submersos pelo núcleo elétrico, que possui além desta, funções de comando e desta forma está presente apenas nos cabos umbilicais.

- **Cabos Óticos (*Optic Cable*):** Da mesma forma que o núcleo elétrico, este elemento é uma alternativa para o controle de equipamentos.

- **Mangueiras Hidráulicas (*Hose*):** Esses componentes são responsáveis pelo controle dos equipamentos submersos através da injeção de fluidos como óleos ou fluidos químicos, sendo um componente presente apenas nos cabos umbilicais.

Com estes componentes é possível o projeto dos mais variados tipos de cabos umbilicais e tubos flexíveis. Para a criação de um novo cabo o usuário deve primeiramente fazer a definição dos componentes que estarão presentes no cabo e seus respectivos materiais. Tendo-se os componentes devidamente configurados no repositório do programa, inicia-se a criação das camadas (*layers*) presentes no cabo desejado, onde devem ser definidas as instâncias dos componentes, determinando-se assim a posição de tal componente na camada projetada.

Durante todo o projeto, é possível observar a secção transversal do cabo na região destinada a visualização bidimensional, assim como interagir diretamente reposicionando instâncias com funções como arrastar e soltar (*drag and drop*), além de funções de visualização básicas como ampliação (*zoom*), arrasto (*pan*) e a seleção de instâncias diretamente pelo desenho.

Capítulo 3. Geração de Sólidos Tridimensionais

Em computação gráfica, existe um grande desafio na criação de superfícies curvas, pois computadores não possuem a capacidade de processamento contínuo, trabalhando sempre no espaço discreto. Devido a este fato, superfícies curvas são produzidas pela criação de diversos polígonos planos dispostos lado a lado, de forma a criar a sensação de curvatura desejada.

Para a criação das superfícies, é necessária a geração de uma malha contendo os diversos vértices que a irão compor. Em casos simples, estes vértices podem ser calculados manualmente, porém, com o aumento do número de pontos, esta técnica se torna inviável, sendo necessária a criação de um algoritmo para a geração da malha desejada.

O processo de geração de polígonos para compor uma superfície é conhecido como ‘*tesselation’*, e para o caso específico onde os polígonos são todos triângulos, a palavra ‘*triangularization’* (triangularização) é mais apropriada.

A utilização de triângulos para tal aproximação é o método mais utilizado ao se fazer a geração de malhas, uma vez que a unidade de processamento gráfico (GPU) das placas de vídeo possuem desempenho otimizado para cálculos com esta geometria. Existem diversas técnicas para tal fim, sendo os principais delas descritas a seguir.

3.1. Geração de sólidos por parametrização

Ao criar uma malha, para se ter controle de todos os pontos e também flexibilidade de fazer uma malha tão refinada quanto se desejar, uma técnica muito utilizada é a parametrização do sólido. Equações paramétricas são equações que representam pontos de uma linha ou uma superfície através de uma ou mais variáveis, chamadas de parâmetros.

Tomando como exemplo uma circunferência bidimensional de raio r centrada em (x_0, y_0), a equação analítica que a representa é dada por:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \quad 3.1$$

No entanto, equações analíticas não permitem saber quais são os pontos da circunferência, pode-se apenas saber se determinado ponto pertence ou não a ela. As equações paramétricas que representam a mesma circunferência são dadas por:

$$x(t) = x_0 + r\cos(2\pi t) \quad 3.2$$

$$y(t) = y_0 + r\,\mathrm{sen}(2\pi t) \quad 3.3$$

Nota-se nessas equações que é utilizada a variável t, que pode variar de 0 a 1 para descrever a circunferência inteira. Ao se variar o parâmetro t é possível se determinar os pontos $\{x(t), y(t)\}$ que pertencem à circunferência, podendo-se obter tais pontos com o refinamento que se desejar.

Ao se descrever superfícies tridimensionais, são utilizados dois parâmetros. No caso de uma esfera, os raios das circunferências que a compõem variam de acordo com a variação da coordenada z, conforme mostra a Figura 3.1. Nesta figura, a coordenada z é facilmente determinada sendo correspondente a $z(\theta, \Phi) = R\sin\Phi$, já o raio de cada anel é dado por $r(\theta, \Phi) = R\cos\Phi$. Desta forma, as equações parametrizadas da esfera são dadas por:

$$x(\theta, \Phi) = x_0 + R\cos\Phi\cos(\theta) \quad 3.4$$

$$y(\theta, \Phi) = y_0 + R\cos\Phi\,\mathrm{sen}(\theta) \quad 3.5$$

$$z(\theta, \Phi) = z_0 + R\,\mathrm{sen}\,\Phi \quad 3.6$$

Onde θ pode variar entre $0 < \theta < 2\pi$, e Φ pode variar entre $-\pi/2 < \Phi < \pi/2$.

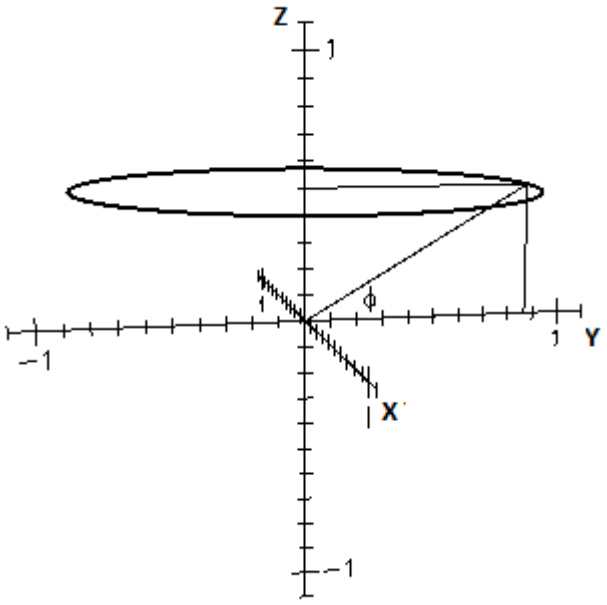

Figura 3.1 - Ângulos para a parametrização da esfera – Extraído de (3)

Tendo as equações paramétricas definidas, deve-se criar ordenadamente todos os vértices para que a aproximação da superfície curva seja adequada e gere o efeito desejado. A criação destes vértices pode ser feita definindo-se incrementos para as variáveis paramétricas, sendo que quanto menor os incrementos, melhor a aproximação feita. Neste ponto, a equação paramétrica se mostra de extrema importância, pois permite esta variação do tamanho da malha.

Uma vez gerados todos os vértices, estes devem ser unidos de forma a se definir os triângulos que irão compor a superfície. A sequência de definição destes vértices é essencial, pois definirá a direção da normal e desta forma, a face frontal e o verso de cada triângulo. A Figura 3.2 mostra o exemplo de um cilindro, onde sua superfície foi aproximada por triângulos.

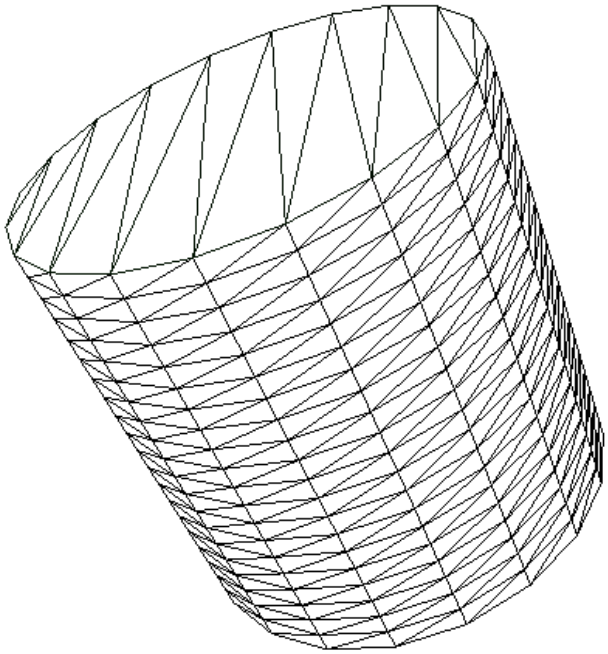

Figura 3.2 - Exemplo de malha de um cilindro utilizando elementos triangulares

A parametrização dos sólidos é uma maneira muito eficiente de se gerar malhas, permitindo assim um grande controle de malha, podendo ser utilizada em diversos sólidos, como cilindros, cones, cubos, dentre outros, sendo necessário apenas encontrar a equação paramétrica que os representa.

3.2. Geração de sólidos por operações booleanas

Para determinadas superfícies, pode não ser possível encontrar uma equação que as represente adequadamente, ou estas equações podem apresentar um nível de complexidade muito elevado. Neste caso, deve ser verificada a possibilidade de se utilizar sólidos primitivos, e através de operações booleanas, compor a superfície desejada, uma vez que existem algoritmos para a determinação da interseção de superfícies.

A Figura 3.3 mostra um exemplo de operações booleanas com um cilindro e uma esfera.

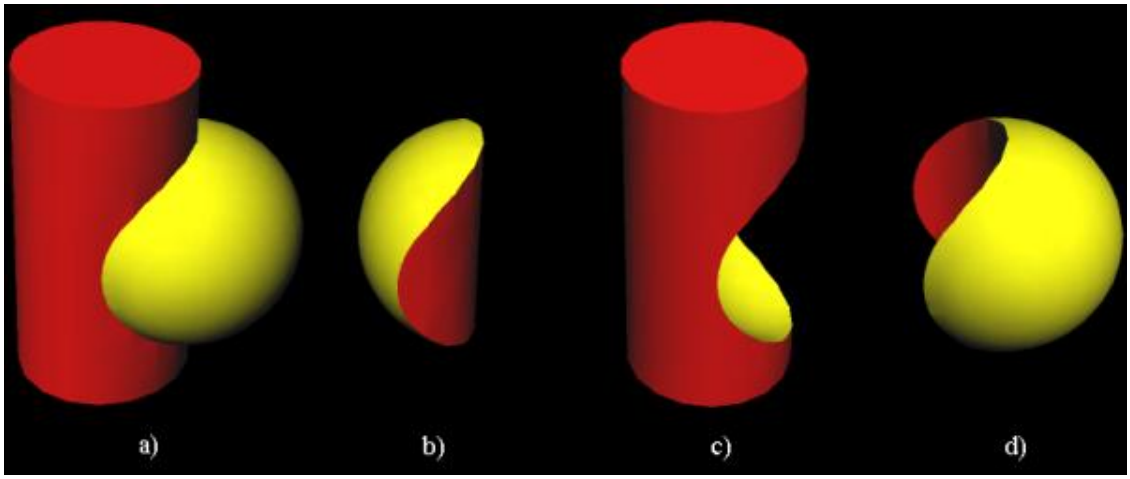

Figura 3.3 - Operações booleanas com sólidos – Extraído de (4)

As operações booleanas são divididas basicamente em:

- União: Se dá pela junção de dois sólidos, que geram apenas um. Todas as partes de ambos os sólidos são preservadas. Representa o caso ‘a’ da figura acima.
- Intersecção: O sólido gerado corresponde apenas a região comum aos dois sólidos primitivos. Representa o caso ‘b’ da figura acima.
- Subtração: Neste caso, o sólido gerado é definido por um sólido inicial subtraído da região onde ocorre intersecção com um segundo sólido. Este caso é representado em ‘c’ e em ‘d’ na figura, onde o cilindro é subtraído da esfera e a esfera é subtraída do cilindro, respectivamente.

3.3. Geração de sólidos por revolução

A técnica de geração de sólidos por revolução (*revolve*) é muito utilizada quando se deseja a criação de objetos axissimétricos. Para a criação de um objeto tridimensional, é necessário um perfil bidimensional, o qual será rotacionado em torno de um eixo dado, produzindo assim o efeito desejado.

A Figura 3.4 mostra alguns exemplos de sólidos gerados por revolução. Como se pode observar, está é uma técnica bastante simples que permite produzir uma grande variedade de objetos tridimensionais.

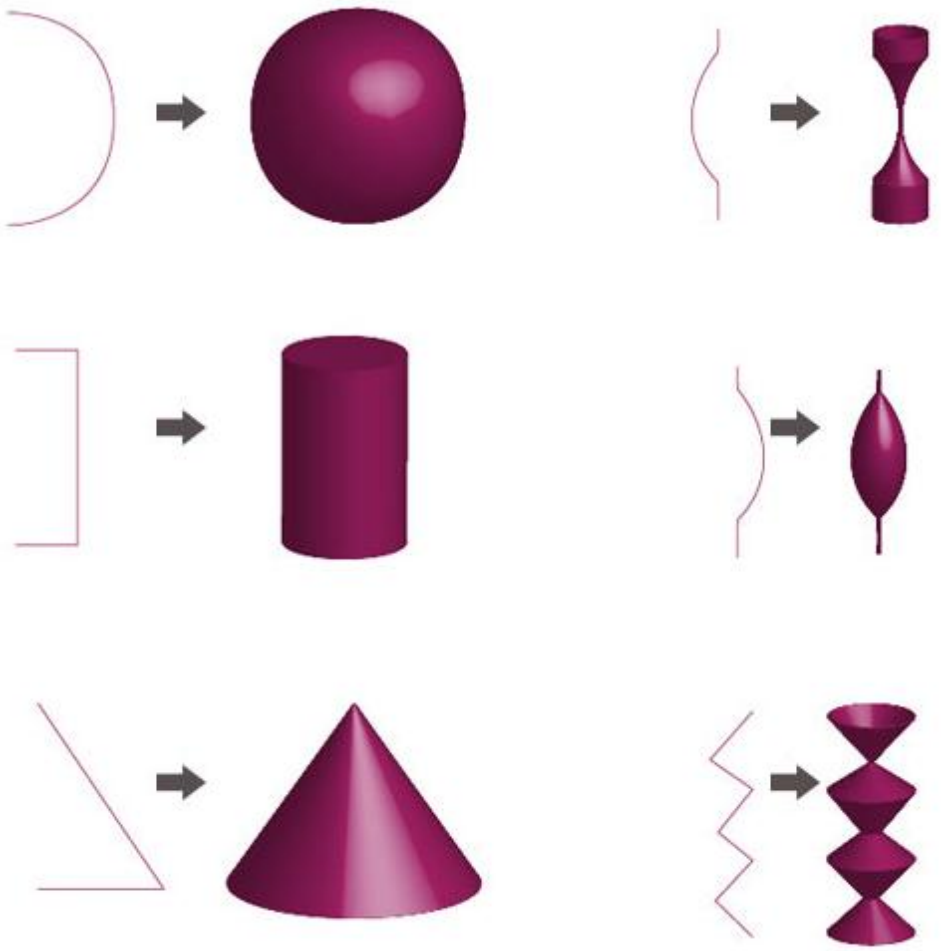

Figura 3.4 - Exemplos de sólidos gerados por revolução – Extraído de (5)

Para que um sólido seja gerado por esta técnica, deve-se primeiramente ter posse dos pontos que definem o perfil a ser rotacionado. Posteriormente, é necessário que seja definido um centro de rotação e um vetor em torno do qual será feita a operação, podendo deixar como opção do usuário sua seleção, ou uma seleção automática, com a origem como centro de rotação e o vetor $(0,0,1)$ como sua orientação, por exemplo. Com isto, o próximo passo necessário é a definição do ângulo de incremento, o qual definirá dois perfis consecutivos, para posterior união.

Assumindo-se que um ponto (x, y, z) é pertencente ao perfil, e este deve ser rotacionado em torno do ponto (x_0, y_0, z_0) direcionado pelo vetor (0,0,1), com um incremento angular de $d\theta$, suas novas coordenadas serão:

$$x' = (x - x_0) \cdot \cos(d\theta) - (y - y_0) \cdot \text{sen}(d\theta) \quad 3.7$$

$$y' = (x - x_0) \cdot \text{sen}(d\theta) + (y - y_0) \cdot \cos(d\theta) \quad 3.8$$

$$z' = z \quad 3.9$$

Fazendo-se esta rotação sucessivas vezes, serão gerados diversos perfis consecutivos, bastando, portanto, a união dos mesmos para a geração do sólido.

É importante observar que a qualidade do objeto gerado é inversamente proporcional ao ângulo de incremento, pois ângulos grandes podem produzir objetos grosseiros, uma vez que perfis consecutivos são conectados com superfícies planas (geralmente, triângulos).

3.4. Geração de sólidos por extrusão e arrasto

Como o próprio nome já sugere, estas duas técnicas consistem na geração de um sólido tridimensional pelo arrasto de uma imagem plana, sendo que no caso da extrusão, tem-se como caminho uma reta perpendicular à imagem plana, enquanto que o arrasto pode ser feito ao longo de uma curva qualquer. Bastante semelhante à técnica de revolução apresentada anteriormente, o objeto tridimensional é formado pela união de consecutivos perfis.

Para a geração de um sólido por extrusão, basta um perfil bidimensional, e a distância pelo qual ele deve ser arrastado. Desta forma, uma transformação de translação é aplicada igualmente a todos os pontos do perfil, e o sólido desejado será gerado pela união destes perfis. Dependendo do caso, pode ser interessante subdividir a distância total de deslocamento

em pequenos incrementos, gerando diversos perfis consecutivos que devem ser unidos de forma adequada. A Figura 3.5 mostra um exemplo de um sólido gerado por esta técnica.

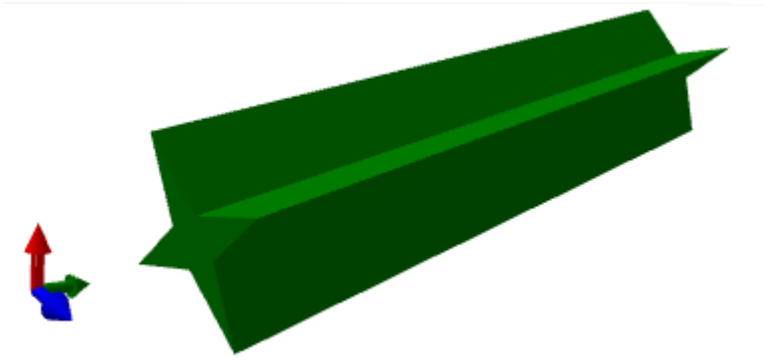

Figura 3.5 - Exemplo de sólido gerado por extrusão

A técnica de arrasto (*sweep*) possui o conceito bastante semelhante à extrusão, porém permite uma maior flexibilidade, de modo que o perfil não precisa necessariamente estar perpendicular ao caminho a ser percorrido, além da possibilidade de se construir um caminho genérico. Adicionalmente, podem ser aplicados outros efeitos ou transformações, como rotação do perfil, escala, distorção, entre outras, bastando que sejam definidos os seus parâmetros.

Para a aplicação desta técnica, é necessária a definição do perfil a ser arrastado, assim como a curva pela qual será efetuada a operação. Adicionalmente, caso desejado, serão necessárias as funções de variação de cada parâmetro, as quais definem as transformações a serem aplicadas durante a operação. A Figura 3.6 mostra um exemplo de sólido gerado por esta técnica.

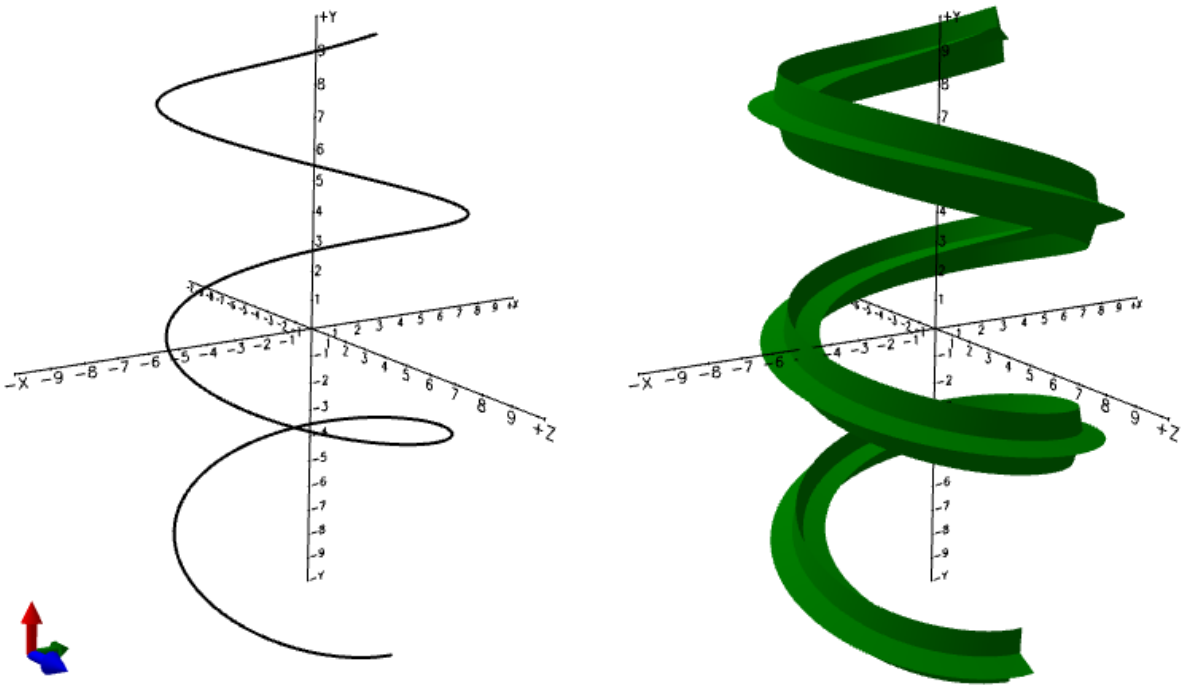

Figura 3.6 - Caminho percorrido (esquerda) e sólido gerado por arrasto (direita)

Durante seu percurso, tem-se como restrição essencial que o ângulo formado entre a normal do perfil e a tangente da curva seja constante (ou, caso desejado, que sua variação siga uma função dada). A Figura 3.7 mostra como esta restrição deve ocorrer para um caso bidimensional.

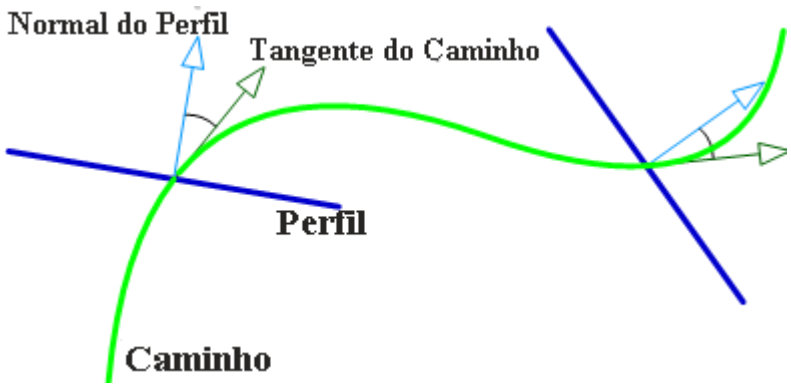

Figura 3.7 - Exemplo do ângulo entre normal e tangente da curva constantes

A implementação desta restrição pode ser feita de duas maneiras:

- Rotação do perfil em torno do centro de rotação da curva:

- Encontrar o centro de rotação da curva;
- Fazer o produto vetorial entre os vetores que ligam o centro de rotação da curva com a posição atual e com a posição final, assim como o cálculo do ângulo entre eles;
- Aplicar a rotação no perfil, utilizando-se como centro de rotação o mesmo que o da curva, e eixo de rotação o vetor encontrado anteriormente.

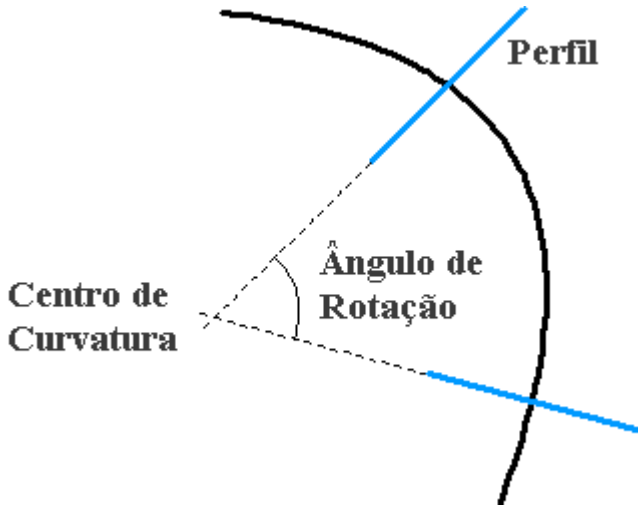

Figura 3.8 - Exemplo de rotação de um perfil centrado no centro de curvatura da curva

- Rotação do perfil em seu próprio centro e aplicação de uma translação para correção da posição:
  - Calcular o produto vetorial e o ângulo dos vetores tangentes à curva na posição atual e na próxima posição desejada.
  - Aplicar uma rotação ao perfil, com centro em sua própria origem, e como eixo de rotação o vetor encontrado anteriormente.
  - Aplicar uma transformação de translação de modo a colocar o perfil na nova posição.

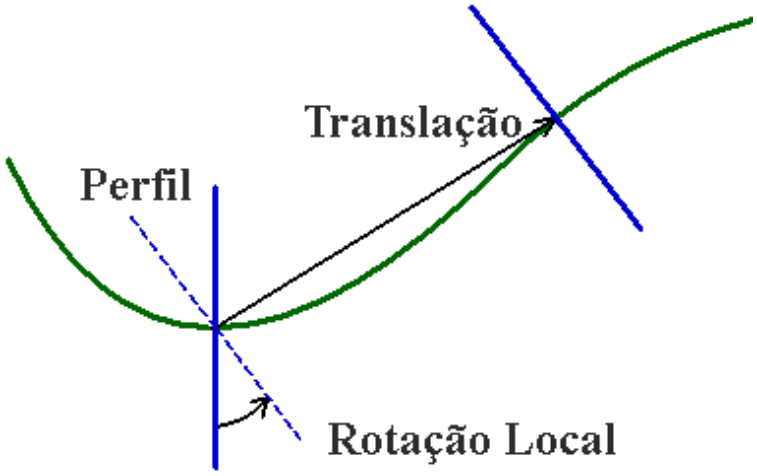

Figura 3.9 - Exemplo de rotação do perfil em seu próprio centro e posterior translação

Apesar de a primeira técnica parecer mais prática, uma vez que é necessário aplicar apenas uma transformação ao perfil, ela se demonstra restrita, não podendo ser utilizada para casos onde o centro de rotação se encontra no infinito (retas), ou curvas com diversos centros de rotação.

Da mesma forma que as técnicas de revolução e extrusão apresentadas anteriormente, pode ser necessária a definição de incrementos ao percorrer o caminho desejado. Assumindo uma curva s e um incremento Δs, têm-se:

$$\vec{s}(0) = (x_{s0}, y_{s0}, z_{s0})$$
$$\vec{s}(\Delta s) = (x_{s1}, y_{s1}, z_{s1}) \quad 3.10$$

E os seus respectivos vetores tangentes:

$$\vec{s}'(0) = \left(x'_{s0}, y'_{s0}, z'_{s0}\right)$$
$$\vec{s}'(\Delta s) = (x'_{s1}, y'_{s1}, z'_{s1}) \quad 3.11$$

Pode-se aplicar uma rotação ao perfil, com o seguinte eixo de rotação e ângulo:

$$Eixo = \vec{s}'(0) \times \vec{s}'(\Delta s) \quad 3.12$$

$$\theta = \hat{A}nguloEntre(\vec{s}'(0)\ ;\ \vec{s}'(\Delta s)) \quad 3.13$$

Posteriormente, a seguinte translação deve ser aplicada:

$$(\Delta x, \Delta y, \Delta z) = \vec{s}(\Delta s) - \vec{s}(0) = (x_{s1} - x_{s0}, y_{s1} - y_{s0}, z_{s1} - z_{s0}) \quad 3.14$$

Adicionalmente, caso tenha sido definido qualquer outra transformação a ser aplicada ao perfil, esta deve ser feita nesta etapa do processo.

O sólido desejado será gerado pela aplicação destes passos sucessivas vezes, para cada ponto do perfil ao longo do caminho definido, e a posterior união entre perfis consecutivos.

Capítulo 4. Transformações e Projeções

Em computação gráfica, muitas vezes é necessária a utilização de transformações, seja para aplicação de efeitos de animação, para facilitar criação de objetos ou até mesmo para o simples posicionamento dos mesmos em uma cena.

Em seguida, será descrito quais são as transformações mais utilizadas, assim como a técnica empregada para sua modelagem.

4.1. Transformações Básicas

Os principais tipos de transformações existentes são: translação (*translate*), rotação (*rotate*), escala (*scale*) e cisalhamento (*shear*)

A transformação de translação consiste em mover um ponto (ou objeto) no espaço. Para tal, deve-se apenas acrescentar um deslocamento em suas coordenadas.

Para um ponto P, ao aplicar uma translação T, tem-se:

$$P = [x \quad y \quad z];\ T = [\Delta x \quad \Delta y \quad \Delta z];\ P' = [x' \quad y' \quad z'] \tag{4.1}$$

$$P' = P + T \tag{4.2}$$

$$[x' \quad y' \quad z'] = [x + \Delta x \quad y + \Delta y \quad z + \Delta z] \tag{4.3}$$

A Figura 4.1 mostra um exemplo desta transformação.

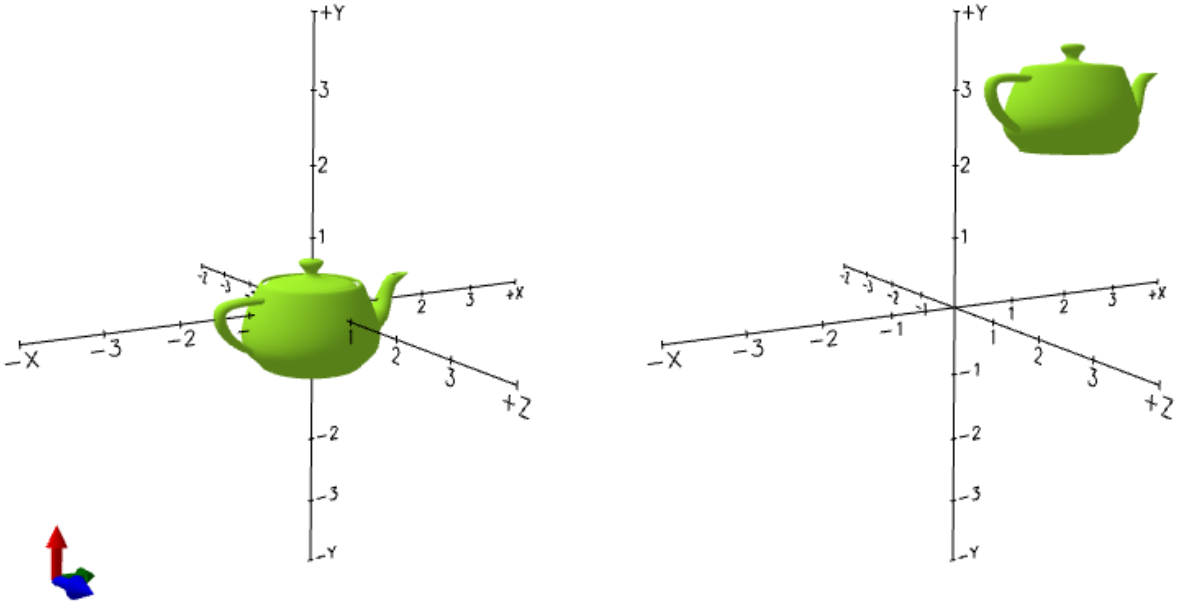

Figura 4.1 - Exemplo de transformação de translação

A transformação de rotação consiste em rotacionar um ponto (ou objeto) em torno de um centro de rotação e um eixo de rotação por um determinado ângulo.

Dado um ponto $P = (x, y, z)$, ao aplicarmos uma rotação em torno do eixo Z com centro de rotação na origem e um ângulo θ, teremos:

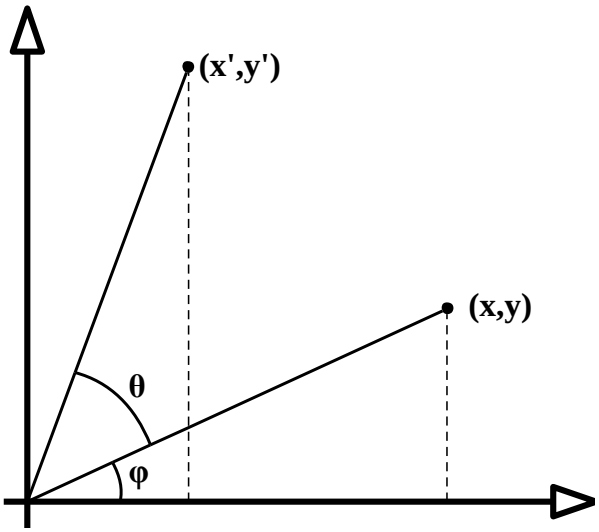

Figura 4.2 - Ângulos utilizado na rotação

$$x' = r \cdot \cos\varphi \cdot \cos\theta - r \cdot \sin\varphi \cdot \sin\theta \qquad 4.4$$

$$y' = r \cdot \cos\varphi \cdot \operatorname{sen}\theta + r \cdot \operatorname{sen}\varphi \cdot \cos\theta \quad (4.5)$$

$$z' = z \quad (4.6)$$

Como:

$$x = r \cdot \cos\varphi \quad (4.7)$$

$$y = r \cdot \operatorname{sen}\varphi \quad (4.8)$$

Têm-se:

$$x' = x \cdot \cos\theta - y \cdot \operatorname{sen}\theta \quad (4.9)$$

$$y' = x \cdot \operatorname{sen}\theta + y \cdot \cos\theta \quad (4.10)$$

Utilizando-se da representação por matriz:

$$P = [x \quad y \quad z]; R = \begin{bmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}; P' = [x' \quad y' \quad z'] \quad (4.11)$$

$$P' = P \cdot R \quad (4.12)$$

Similarmente, as rotações em torno do eixo X e Y possuem as seguintes matrizes de rotação, respectivamente:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \operatorname{sen}\theta \\ 0 & -\operatorname{sen}\theta & \cos\theta \end{bmatrix} \quad (4.13)$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & -\operatorname{sen}\theta \\ 0 & 1 & 0 \\ \operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix} \quad (4.14)$$

Para o caso em que se deseje fazer uma rotação com um centro de rotação diferente da origem, deve-se primeiramente aplicar uma transformação de translação, de modo que o centro de rotação fique na origem, aplicar a rotação, e aplicar novamente uma translação de modo a retornar à posição anterior.

Dado um centro de rotação $C = (c_x, c_y, c_z)$, uma rotação em torno do eixo Z resultaria em:

$$P = [x \quad y \quad z]; P' = [x' \quad y' \quad z'] \tag{4.15}$$

$$R = \begin{bmatrix} \cos\theta & \operatorname{sen}\theta & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}; T = [-c_x \quad -c_y \quad -c_z]; T' = [c_x \quad c_y \quad c_z] \tag{4.16}$$

$$P' = \big((P + T) \cdot R\big) + T' \tag{4.17}$$

Caso seja desejada uma rotação em torno de um eixo genérico, esta operação pode ser decomposta em três rotações, em torno dos eixos conhecidos (X, Y e Z), tomando-se a devida atenção para a ordem das operações, uma vez que a transformação de rotação não é comutativa. A Figura 4.3 mostra um exemplo de uma rotação em torno de um eixo qualquer.

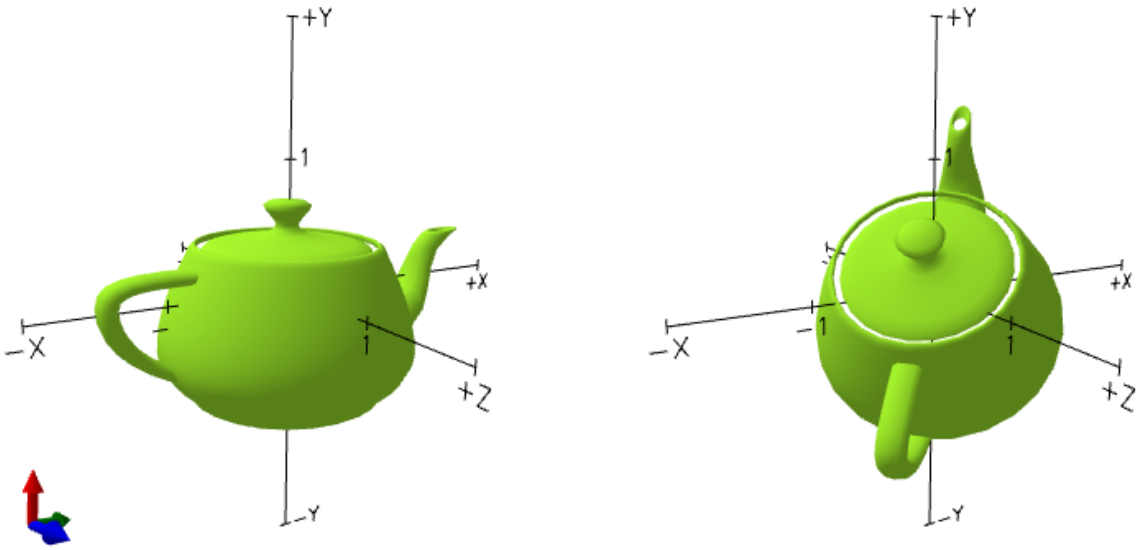

Figura 4.3 - Exemplo de rotação em torno de um eixo qualquer

A transformação de escala consiste em aumentar a distância entre dois pontos (ou aumentar o tamanho de um objeto) de acordo com determinado fator dado.

Dado um ponto P e os fatores de escala s_x, s_y e s_z, têm-se:

$$x' = s_x \cdot x \tag{4.18}$$

$$y' = s_y \cdot y \tag{4.19}$$

$$z' = s_z \cdot z \tag{4.20}$$

E em sua representação matricial:

$$P = [x \quad y \quad z]; S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}; P' = [x' \quad y' \quad z'] \tag{4.21}$$

$$P' = P \cdot S \tag{4.22}$$

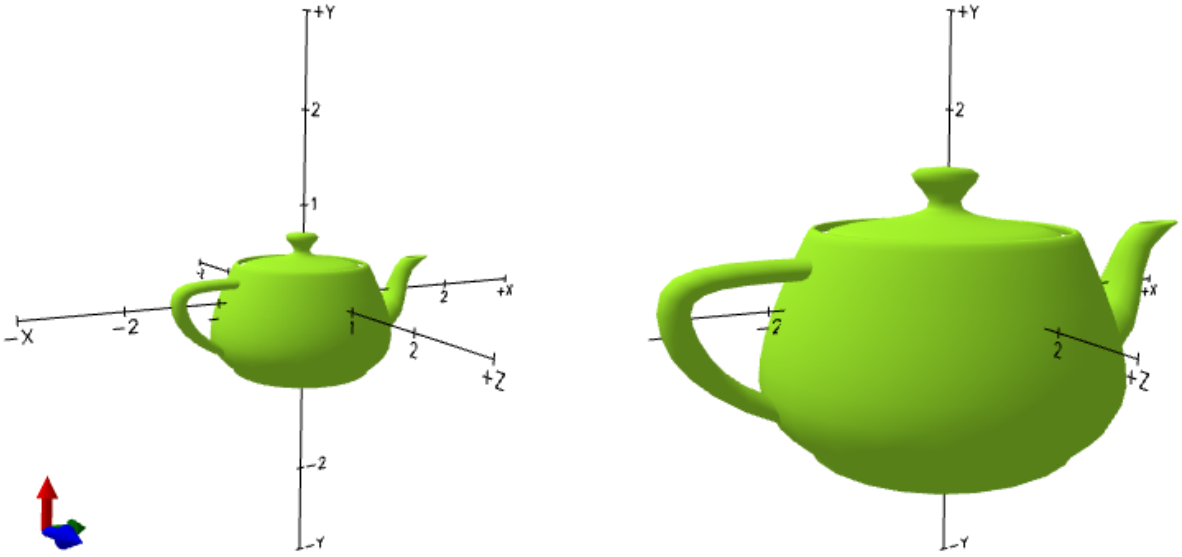

Figura 4.4 - Exemplo de transformação de escala

A transformação de cisalhamento consiste em distorcer um objeto, como se houvesse um deslizamento entre suas camadas.

Aplicando-se uma transformação de cisalhamento com um fator sh na coordenada x, proporcional à coordenada y a um ponto P, têm-se:

$$x' = x + sh \cdot y \tag{4.23}$$

$$y' = y \tag{4.24}$$

$$z' = z \tag{4.25}$$

A Figura 4.5 mostra um exemplo desta transformação.

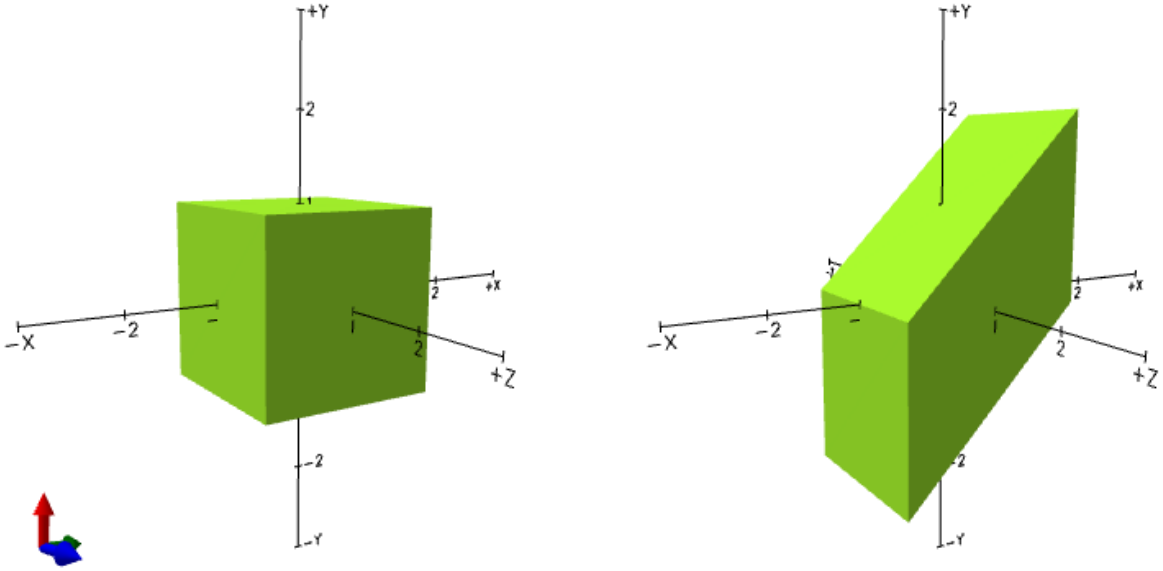

Figura 4.5 - Exemplo de transformação de cisalhamento

Esta transformação também pode ser representada em forma de uma matriz, e a sua forma genérica é:

$$Sh = \begin{bmatrix} 1 & sh_{xy} & sh_{xz} \\ sh_{yx} & 1 & sh_{yz} \\ sh_{zx} & sh_{zy} & 1 \end{bmatrix} \tag{4.26}$$

Onde sh_{mn} é o fator de cisalhamento da coordenada m proporcional à coordenada n.

4.2. Coordenadas Homogêneas

Pode-se observar que as transformações de rotação, escala e cisalhamento podem ser aplicadas a partir das matrizes de transformação apresentadas, porém, a translação é aplicada somando-se a matriz de transformação.

Com o objetivo de se realizar todas as transformações com multiplicações e agrupar diversas transformações em uma única matriz, foi criado o conceito de coordenadas homogêneas, que consiste em acrescentar uma dimensão ao problema em questão.

Desta forma, um ponto P no espaço tridimensional, em coordenadas homogêneas ficaria:

$$P = [x \quad y \quad z \quad 1] \tag{4.27}$$

Com isto, para se realizar uma translação, basta fazer a multiplicação de um ponto em coordenadas homogêneas pela seguinte matriz:

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{4.28}$$

As demais transformações vistas anteriormente ficam, portanto, utilizando-se coordenadas homogêneas:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \operatorname{sen}\theta & 0 \\ 0 & -\operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.29}$$

$$R_y = \begin{bmatrix} \cos\theta & 0 & -\operatorname{sen}\theta & 0 \\ 0 & 1 & 0 & 0 \\ \operatorname{sen}\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.30}$$

$$R_z = \begin{bmatrix} \cos\theta & \operatorname{sen}\theta & 0 & 0 \\ -\operatorname{sen}\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.31}$$

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

$$Sh = \begin{bmatrix} 1 & sh_{xy} & sh_{xz} & 0 \\ sh_{yx} & 1 & sh_{yz} & 0 \\ sh_{zx} & sh_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 4.33$$

4.3. Composição de Transformações

Tendo-se todas as transformações sendo realizadas por multiplicações, é possível a realização de diversas transformações com uma única matriz. Isto é muito vantajoso, pois uma vez que se tem uma grande quantidade de pontos, para os quais é necessária a mesma sequência de transformações, pode ser calculada a matriz de transformação equivalente a todas as transformações e fazer apenas uma multiplicação para cada ponto, reduzindo-se consideravelmente a quantidade de processamento necessário para o cálculo da posição final de todos os pontos.

Por exemplo, se desejamos aplicar uma transformação de escala S e em seguida uma de translação T, podemos calcular a matriz de transformação equivalente M da seguinte maneira:

$$P' = P \cdot S \cdot T = P \cdot (S \cdot T) = P \cdot M \quad 4.34$$

$$M = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ t_x & t_x & t_x & 1 \end{bmatrix} \quad 4.35$$

Quando se deseja aplicar diversas operações de transformação em um ponto, é importante observar que, embora a multiplicação de matrizes seja associativa, a mesma não é comutativa, ou seja, a ordem em que as operações são feitas é importante para o resultado final, uma vez que ao alterarmos a ordem, teremos resultados diferentes.

No caso anterior, se fosse aplicada a transformação de translação antes da de escala, a matriz resultante desta operação seria:

$$M = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ s_x t_x & s_y t_x & s_z t_x & 1 \end{bmatrix} \quad 4.36$$

A Figura 4.6 ilustra a diferença do resultado entre a composição de duas transformações, uma de rotação e outra de translação. A imagem 'A' corresponde a posição original, a imagem 'B' corresponde a aplicação de uma rotação seguida de uma translação, enquanto a imagem 'C' representa a translação aplicada antes da rotação.

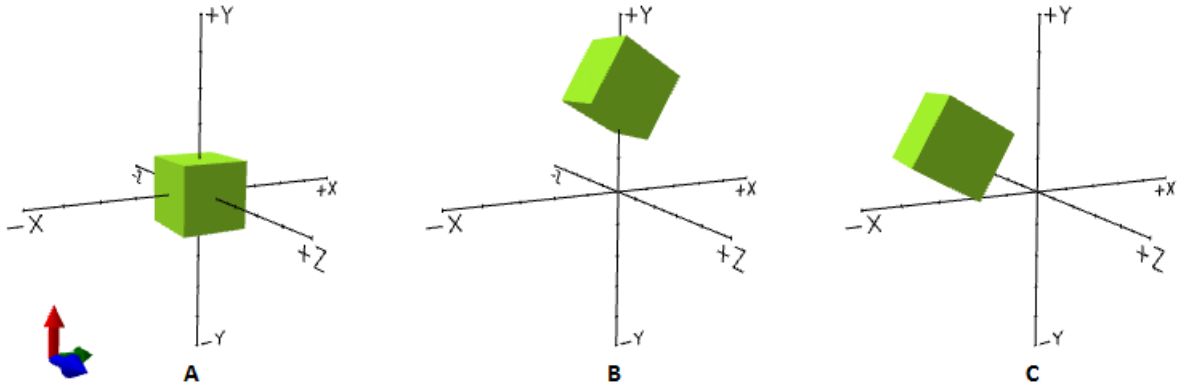

Figura 4.6 - Exemplo de composição de transformações

4.4. Projeções

Foi apresentado como manipular pontos e objetos no espaço tridimensional, no entanto, os monitores atuais não permitem a imersão tridimensional, apenas uma visualização tridimensional planificada, sendo necessária a transformação do sistema de coordenadas 3D para um sistema compatível com os monitores (no caso, o 2D). Este processo é chamado de projeção.

Por exemplo, um cubo, que é definido por seis vértices no espaço tridimensional, deve ser representado no espaço bidimensional, ou seja, todos os seus vértices devem estar no mesmo plano. Um exemplo prático bastante comum no dia a dia é a projeção de sombras dos objetos em uma superfície plana.

Para a realização de uma projeção devem ser considerados três elementos básicos:

- Plano de projeção – plano que conterá a projeção dos pontos tridimensionais;
- Projetantes ou linhas de projeção – retas que passam pelos pontos do objeto em direção ao centro de projeção;
- Centro de projeção – ponto fixo de onde partem as retas projetantes;

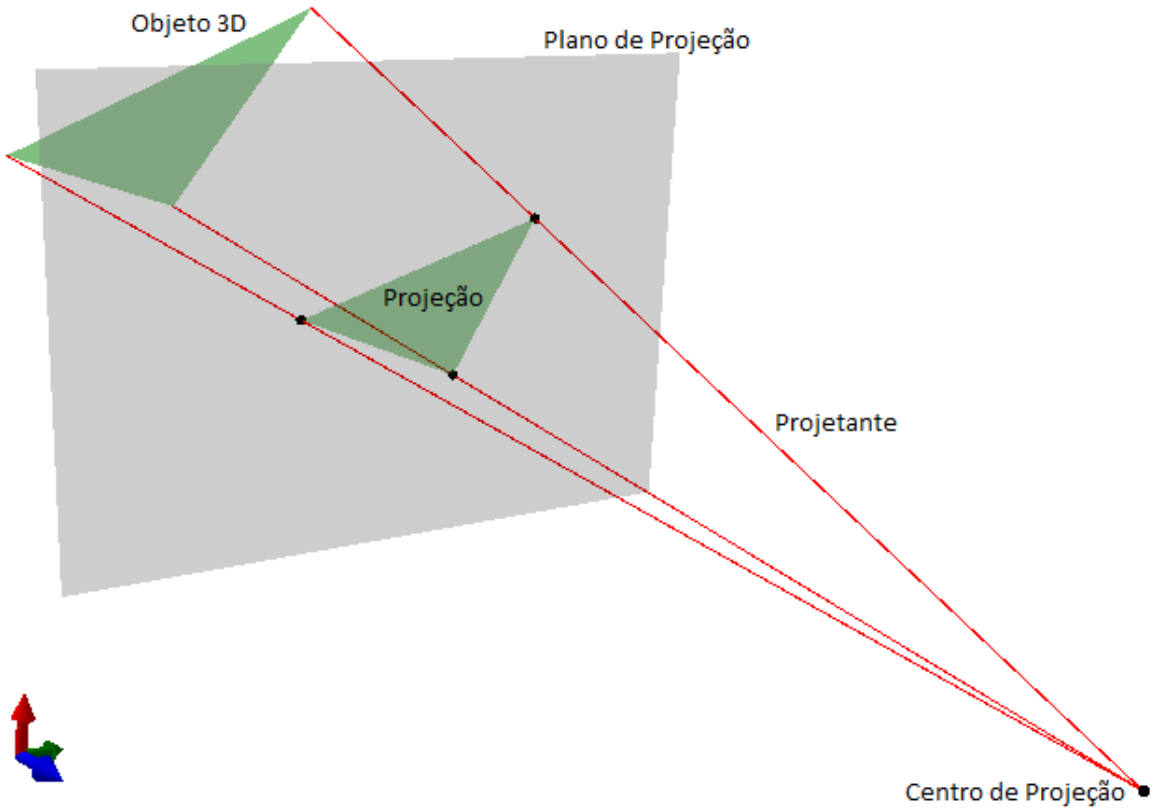

Figura 4.7 - Elementos básicos da projeção

Existem basicamente dois tipos de projeções: a perspectiva e a paralela. Uma projeção perspectiva se assemelha ao processo de formação de imagens em nossos olhos ou câmeras fotográficas, gerando imagens mais realistas. A projeção paralela pode ser dividida ainda em ortogonal, onde as retas projetantes são perpendiculares ao plano de projeção, ou oblíqua, onde as retas projetantes são oblíquas ao plano de projeção. Em ambos os tipos de projeções paralelas, as retas projetantes são paralelas. Esta projeção se assemelha a projeção de uma sombra produzida pelos raios do sol.

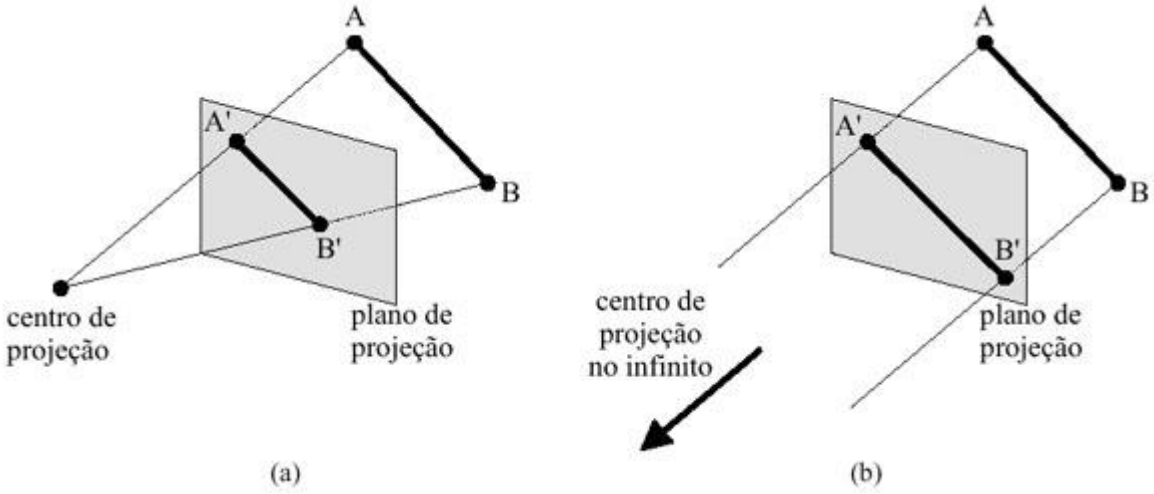

Figura 4.8 - Exemplo de projeção Perspectiva (a) e Paralela (b) – Extraído de (6)

O processo de formação de uma projeção, em computação gráfica, consiste, primeiramente, em normalizar a posição e os eixos da câmera (observador) através da matriz de visualização. Em seguida, deve ser realizada a projeção propriamente dita, realizada através da matriz de projeção.

4.4.1. Matriz de Visualização

A matriz de visualização é responsável por aplicar uma transformação em todos os objetos da cena tridimensional de modo que a câmera fique posicionada na origem, com a sua direção de visualização alinhada com o negativo do eixo Z (em algumas ferramentas de visualização, pode estar alinhada com a direção positiva do eixo Z) e o vetor que aponta para cima alinhado com o eixo Y.

O eixo Z da câmera é o oposto de sua direção de visualização. O eixo Y não é necessariamente o vetor que aponta para cima, uma vez que este vetor não precisa ser perpendicular à direção de visualização, no entanto, o eixo X pode ser calculado por um produto vetorial entre o eixo Z, definido anteriormente, e a direção para cima. Tendo-se os eixos X e Z, o Y pode ser facilmente calculado com mais um produto vetorial entre Z e X. É

importante observar que todos os eixos devem ser normalizados. As equações para o cálculo dos eixos ficam da seguinte forma:

$$Z_{eixo} = -(Direção\ de\ Visualização) \quad 4.37$$

$$X_{eixo} = (Direção\ para\ cima) \times Z_{eixo} \quad 4.38$$

$$Y_{eixo} = Z_{eixo} \times X_{eixo} \quad 4.39$$

A matriz de visualização pode ser calculada de diversas formas, por exemplo, compondo uma transformação de translação, responsável por trazer a câmera para a origem, com três rotações, uma em torno de cada eixo, responsáveis por alinhar os eixos da câmera, o que resultaria em:

$$V = T \cdot R_z \cdot R_y \cdot R_x \quad 4.40$$

4.4.2. Matriz de Projeção

A matriz de projeção é aplicada após a matriz de visualização e é responsável por determinar o que é potencialmente visível na imagem bidimensional, levando em conta as características da câmera, como por exemplo, os planos próximos e distantes, o tamanho do campo de visão, entre outras.

A aplicação da matriz de projeção consiste basicamente em normalizar as coordenadas da cena tridimensional, de forma que apenas as regiões com as seguintes coordenadas são renderizadas na tela:

- Coordenadas X entre -1 e 1;
- Coordenadas Y entre -1 e 1;
- Coordenadas Z entre 0 e 1;

Estando todos os pontos a serem desenhados na tela com suas coordenadas normalizadas, são utilizadas as coordenadas X e Y para seu desenho bidimensional, e a coordenada Z para determinar se um objeto está na frente ou atrás de outro objeto (ao se desenhar um objeto na tela, a coordenada Z de cada ponto é armazenada em um buffer e caso outro objeto esteja na mesma posição, é feita uma comparação entre o valor armazenado e o do objeto atual, e desta forma determinado qual objeto deve ser preservado). A seguir, será apresentado como construir a matriz de projeção para as projeções paralela e perspectiva.

4.4.2.1. Projeção Paralela

Os parâmetros que definem o campo de visualização de uma projeção paralela são:

- Distância do plano mais próximo de visualização;
- Distância do plano mais distante de visualização;
- Largura da visualização;
- Altura da visualização;

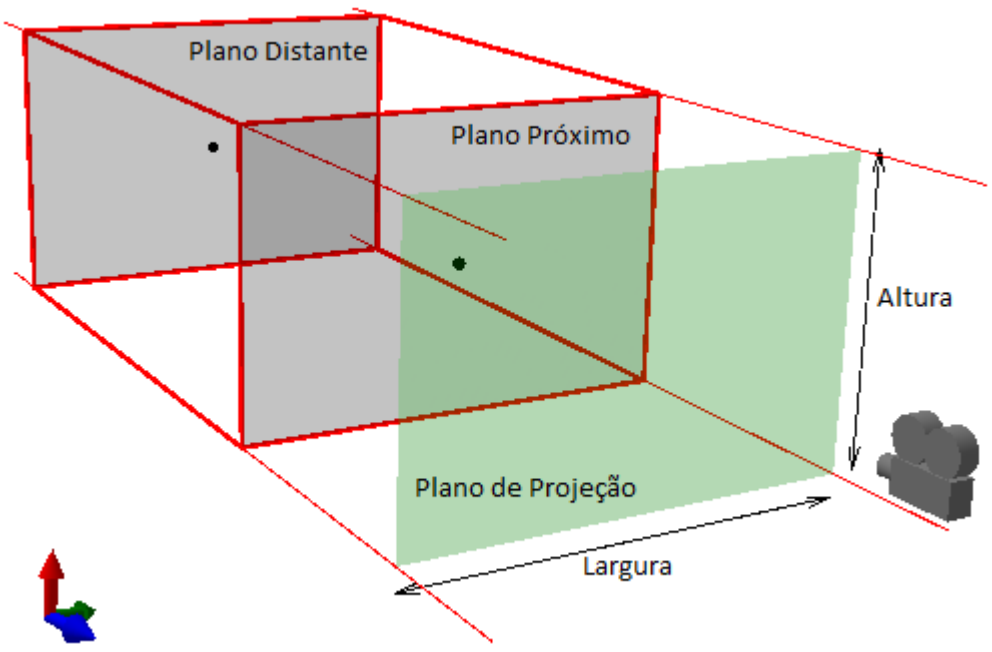

Figura 4.9 - Parâmetro da projeção paralela

Para normalizar as coordenadas conforme exposto anteriormente, é necessário apenas uma transformação de escala com uma translação ao longo de Z. Os fatores de escala para cada eixo e a translação ao longo de Z são dados pelas seguintes fórmulas:

$$s_x = 2/Largura \tag{4.41}$$

$$s_y = 2/Altura \tag{4.42}$$

$$s_z = 1/(PlanoPróximo - PlanoDistante) \tag{4.43}$$

$$\Delta_z = PlanoPróximo \cdot s_z \tag{4.44}$$

Desta forma, têm-se como matriz final para uma projeção ortogonal:

$$P = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & \Delta_z & 1 \end{bmatrix} \tag{4.45}$$

4.4.2.2. Projeção Perspectiva

Os parâmetros que definem o campo de visualização de uma projeção perspectiva são:

- Distância do plano mais próximo de visualização;
- Distância do plano mais distante de visualização;
- Ângulo de abertura de visualização;
- Largura da visualização;
- Altura da visualização;

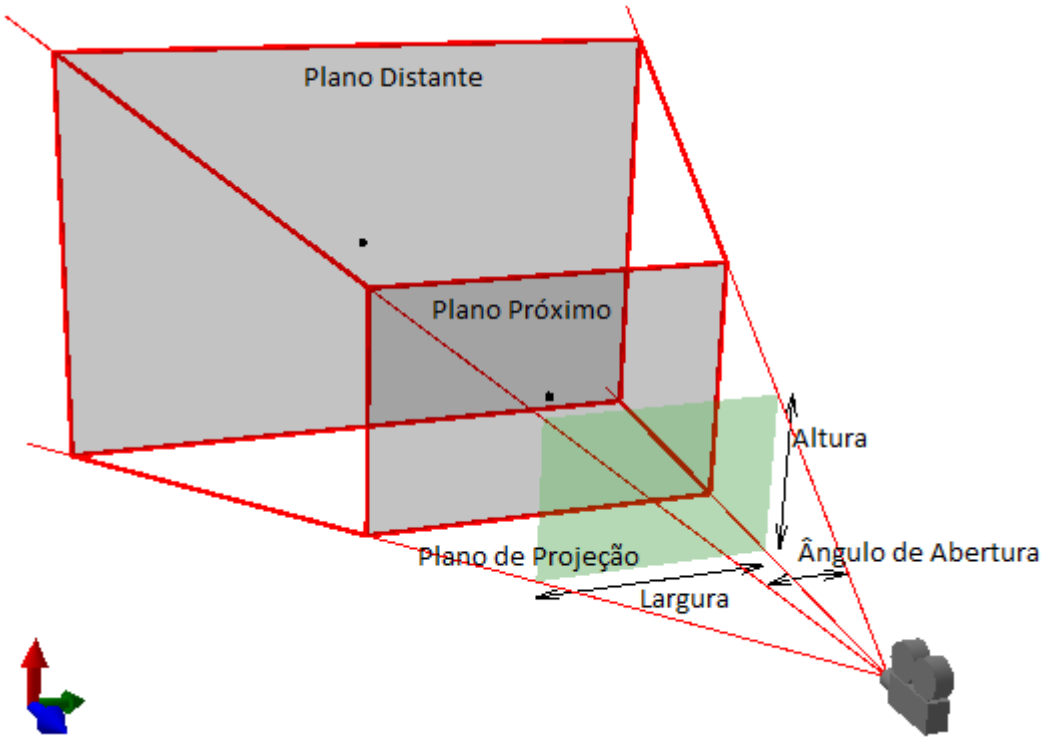

Figura 4.10 - Parâmetros da projeção perspectiva

Tendo-se a largura e altura de visualização, pode-se calcular a razão de aspecto da imagem da seguinte maneira:

$$RazãoDeAspecto = Largura/Altura \tag{4.46}$$

As coordenadas normalizadas podem ser calculadas pelas seguintes fórmulas:

$$x' = \frac{x}{-z \cdot \mathrm{tg}(\hat{A}nguloDeAbertura/2)} \tag{4.47}$$

$$y' = \frac{y \cdot RazãoDeAspecto}{-z \cdot \mathrm{tg}(\hat{A}nguloDeAbertura/2)} \tag{4.48}$$

$$z' = \frac{zDist \cdot z/(zPróx - zDist) + zPróx \cdot zDist/(zPróx - zDist)}{-z} \tag{4.49}$$

Observamos que todas as fórmulas têm em comum o fator $-Z$ no denominador. Fazendo uso de coordenadas homogêneas, lembrando que:

$$[x \quad y \quad z \quad w] \cdot \begin{bmatrix} M11 & M12 & M13 & M14 \\ M21 & M22 & M23 & M24 \\ M31 & M32 & M33 & M34 \\ \Delta_x & \Delta_y & \Delta_z & M44 \end{bmatrix} = [x' \quad y' \quad z' \quad w'] \rightarrow \left(\frac{x'}{w}, \frac{y'}{w}, \frac{z'}{w}\right) \tag{4.50}$$

Fazendo-se $M34 = -1$ e $M44 = 0$, podem ser calculados os demais elementos da matriz da seguinte forma:

$$s_x = 1/\text{tg}(\hat{A}nguloDeAbertura/2) \tag{4.51}$$

$$s_y = RazãoDeAspecto/\text{tg}(\hat{A}nguloDeAbertura/2) \tag{4.52}$$

$$s_z = zDist/(zPróx - zDist) \tag{4.53}$$

$$\Delta_z = zPróx \cdot s_z \tag{4.54}$$

A matriz de projeção final para uma câmera perspectiva resulta em:

$$P = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & -1 \\ 0 & 0 & \Delta_z & 0 \end{bmatrix} \tag{4.55}$$

Capítulo 5. Iluminação e Texturas

Para a compreensão do funcionamento da iluminação e da aplicação de texturas em computação gráfica, é fundamental o entendimento dos diversos padrões para a representação de cores. Serão apresentados apenas os padrões mais utilizados, sendo eles: RGB, CMY, HSV e HLS.

O RGB, por ser o padrão utilizado em equipamentos eletrônicos para a exibição de imagens, como monitores CRT, LCD, Plasma, entre outros, é o mais conhecido e utilizado pelas placas gráficas para seu processamento de cores, além do fato do olho humano também utilizar este padrão para o reconhecimento de cores.

Seu nome é originário das cores principais que o compõem, vermelho (*Red*), verde (*Green*) e azul (*Blue*). Sendo um modelo de cores aditivo, suas cores primárias são misturadas com diferentes intensidades, de forma a gerar uma grande variedade de cores.

Cada uma de suas cores pode variar entre os valores 0 e 255, onde 0 significa ausência da referida cor e 255 significa sua máxima intensidade. Esta variação foi escolhida por ser a quantidade máxima de informação disponível para armazenamento em um byte ($1\ byte = 2^8\ bits = 256\ bits$), desta forma, utilizando-se um byte para cada cor (três bytes no total), têm-se a possibilidade de representar mais de 16 milhões de cores ($256^3 = 16.777.216\ cores\ possíveis$) com este padrão.

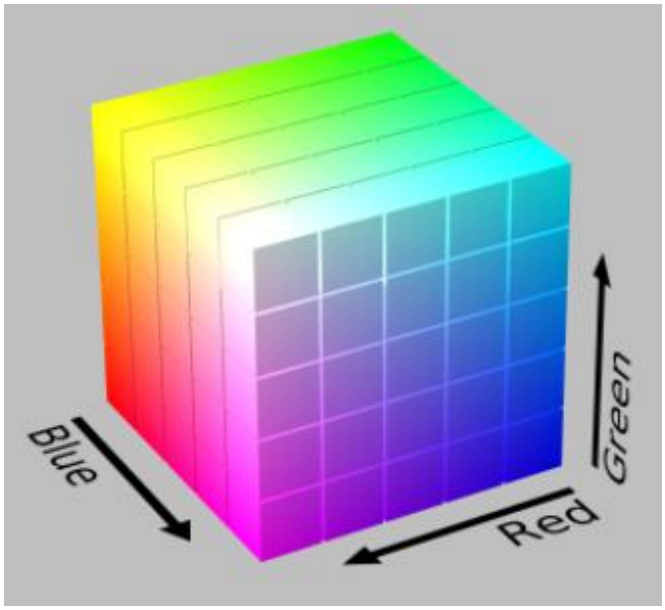

Figura 5.1 - Espaço de cores RGB – Extraído de (7)

Neste modelo de cores, tem-se que a mistura entre vermelho e azul resulta em magenta, entre azul e verde resulta em ciano e entre verde e vermelho resulta em amarelo. Para se obter a cor branca, devem ser adicionadas todas as cores em sua intensidade máxima, assim como para a cor preta, é necessária a ausência de todas as cores.

O padrão de cores CMY é muito parecido com o RGB, porém, utiliza as cores ciano (*Cyan*), magenta (*Magenta*) e amarelo (*Yellow*) como cores primárias. Este é um padrão de cores subtrativo, onde utiliza a luz refletida para exibir a cor. Outra variação deste padrão é o CMYK, onde possui ainda a cor preta como cor primária, sendo o padrão utilizado em impressoras.

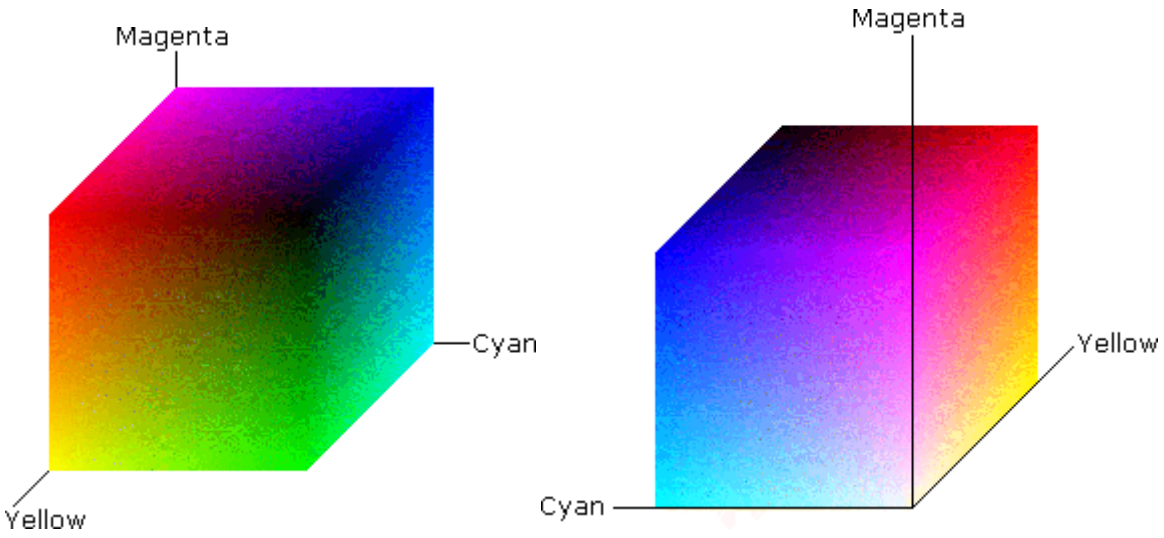

Figura 5.2 - Espaço de cores CMY – Extraído de (8)

Neste padrão, suas cores primárias podem varia de 0 a 100, onde 0 é a ausência da cor, e 100 representa a cor em sua máxima intensidade. Em oposição ao padrão RGB, este padrão utiliza a ausência de todas as cores para representar o branco, e todas as cores em sua máxima intensidade para representar o preto.

Os padrões HSV e HSL são os sistemas de representação de cores cilíndricos mais comuns, onde basicamente ocorre um rearranjo da geometria do RGB, sendo muito utilizado em seletores de cores.

A sigla HSV significa tom (*Hue*), saturação (*Saturation*) e valor (*Value*), enquanto que HSL significa tom (*Hue*), saturação (*Saturation*) e iluminação (*Lightness*). Ambos padrões partem do princípio de que todas as cores reais são originárias de uma única cor, definida por seu tom, criando-se suas variações de acordo com variações de saturação e valor (ou iluminação, para o HSL).

O tom é portanto o que define a cor básica, e é medido em graus, variando de 0 a 360, onde 0 representa vermelho, 60 representa amarelo, 120 representa verde, 180 representa ciano, 240 representa azul e 300 representa magenta.

A saturação representa a variação de uma mesma tonalidade variando de branco para a cor pura. Este valor é medido em porcentagem, onde quanto maior sua porcentagem, mais pura será a cor representada.

O valor representa a intensidade da cor, variando em porcentagem, a cor varia de preto (0%) até a cor pura (100%).

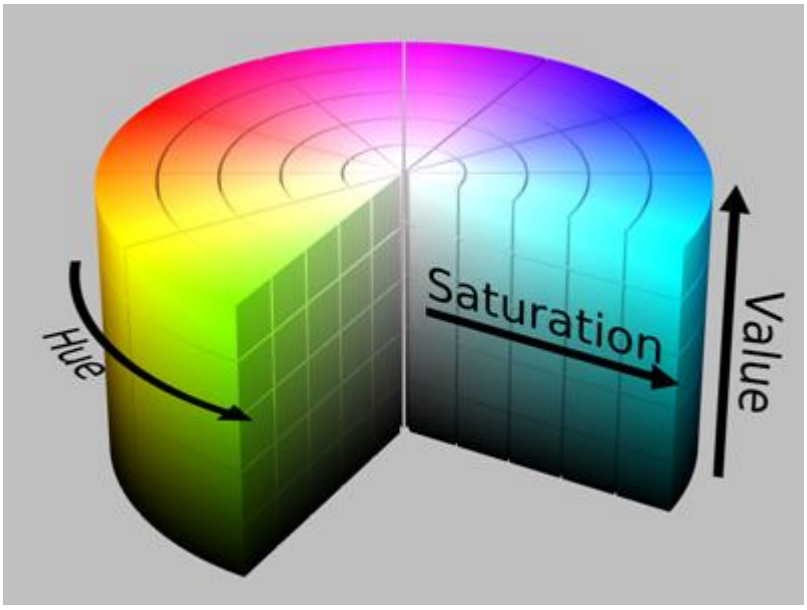

Figura 5.3 - Espaço de cores HSV – Extraído de (7)

Semelhante ao parâmetro valor do padrão HSV, o parâmetro iluminação representa a quantidade de preto ou branco na cor, onde aumentando-se a iluminação, aumenta-se a quantidade de branco misturada a cor. Enquanto no HSV se tem o preto e a cor pura ao se variar o valor, no HSL temos o preto e o branco na variação da iluminação, não existindo, portanto, uma posição em que se possa obter a cor pura. Este parâmetro também é medido em porcentagem, e pode variar entre 0 e 100.

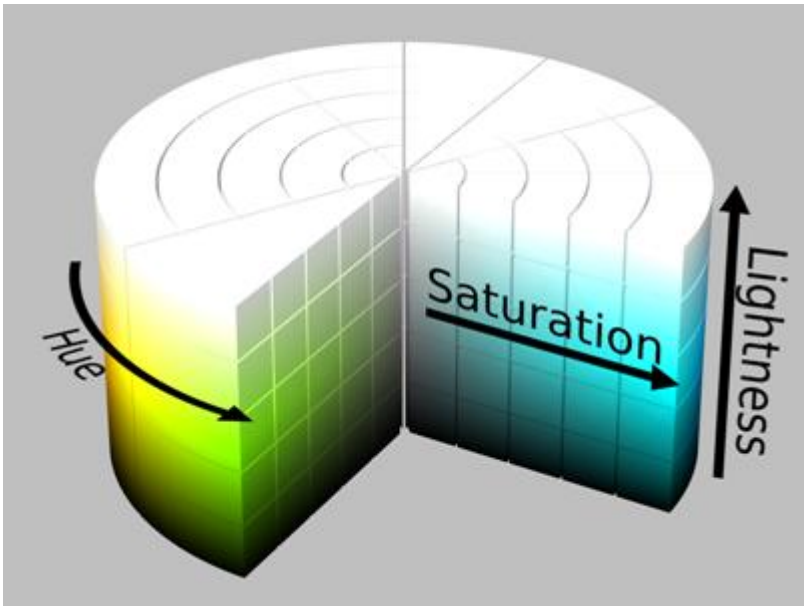

Figura 5.4 - Espaço de cores HSL – Extraído de (7)

5.1. Técnicas de Iluminação

Em computação gráfica, não é possível tratar a iluminação da forma que acontece no mundo real, uma vez que seria muito custoso computacionalmente, pois um único raio de luz pode refletir inúmeras vezes no mesmo objeto, além de existir a interação entre os objetos da cena. Com isso, foram criados modelos para simular, de forma simplificada, a interação da luz com os objetos de uma cena tridimensional.

Os modelos podem ser divididos, basicamente, em dois grupos:

- Reflexão Local;
- Reflexão Global;

Para a aplicação dos modelos, deve ser levada em conta a fonte luminosa, de forma a obter a intensidade do raio de luz que chega ao objeto.

5.1.1. Principais Tipos de Fontes Luminosas

As fontes luminosas são responsáveis por alterar a área e o modo como os raios de luz incidem em um determinado objeto, assim como sua intensidade, sendo fundamentais para a determinação da cor resultante calculada pelos modelos de iluminação, uma vez que este se baseiam na intensidade e direção do raio incidente.

Os principais tipos de fontes luminosas são descritos a seguir.

5.1.1.1. Ambiente

Esta fonte luminosa simula uma iluminação global, onde todos os objetos são atingidos por uma mesma intensidade, independentemente de seu posicionamento.

É utilizada quando se deseja que mesmo objetos que não estão expostos a nenhuma fonte de luz sejam iluminados, como ocorre no caso de um objeto iluminado que, ao refletir os raios de luz, ilumina objetos que estejam na sombra.

A iluminação ambiente não possui direção e é definida somente por uma cor.

5.1.1.2. Direcional

A fonte luminosa direcional trata a luz como vindo sempre de uma mesma direção, independente do ponto de incidência. Esta fonte luminosa se assemelha à iluminação produzida pelo sol, onde todos os raios são paralelos.

Para a definição desta iluminação utiliza-se um vetor indicando sua direção, além da cor da luz que é emitida pela fonte.

A Figura 5.5 ilustra como ocorre a distribuição dos raios de luz neste tipo de fonte luminosa.

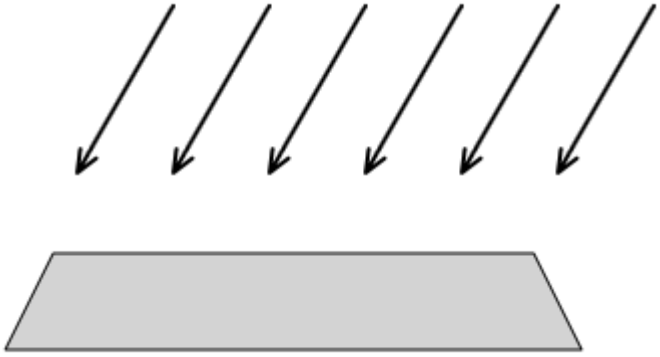

Figura 5.5 - Fonte luminosa direcional – Extraído de (3)

5.1.1.3. Pontual

A fonte de luz pontual nada mais é do que a simulação da iluminação produzida por uma lâmpada. Neste tipo de fonte, a luz é emitida igualmente em todas as direções, como pode ser observado na Figura 5.6.

Para a definição desta iluminação, é necessário apenas um ponto, que será a origem dos raios luminosos, além da cor da luz que é emitida pela fonte.

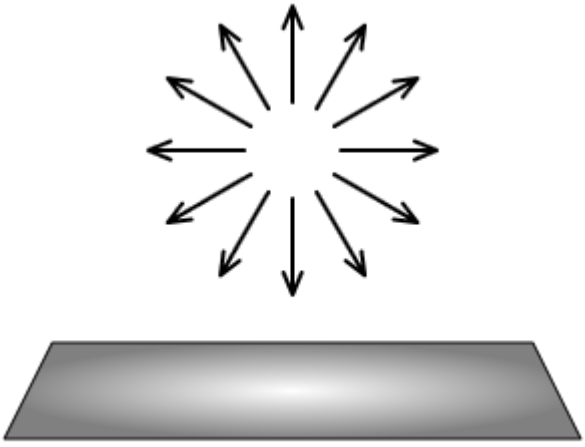

Figura 5.6 - Fonte luminosa posicional – Extraído de (3)

5.1.1.4. Spot Light

Uma fonte de luz do tipo Spot Light é teoricamente bastante semelhante à fonte de luz posicional, no entanto, possui uma região cônica delimitada para emitir luz, não a emitindo

em todas as direções. A Figura 5.7 ilustra como ocorre a iluminação de uma superfície por esta fonte luminosa.

Para a definição deste tipo de iluminação, deve-se ter, além dos parâmetros necessários para a fonte de luz pontual, uma direção principal e o ângulo de abertura do cone que está em torno desta região.

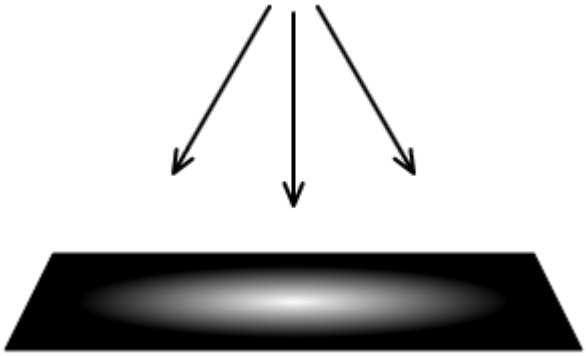

Figura 5.7 - Fonte luminosa tipo 'Spot Light' – Extraído de (3)

5.1.2. Reflexão Local

Um modelo de iluminação com reflexão local considera apenas a luz que atinge o observador ao refletir em apenas um objeto, ou seja, neste modelo de iluminação não se tem a interação entre os objetos, sendo impossível gerar sombras de uma maneira direta, por exemplo, por se tratar cada objeto separadamente.

Para o cálculo da cor em um determinado ponto, normalmente utiliza-se o método de Phong, uma vez que este método propõe aproximações que permitem um efeito bastante próximo do real e esta implementado em hardware nas placas gráficas atualmente. Neste método, a luz é dividida em três componentes:

- Ambiente: Esta componente é utilizada para simular um efeito de iluminação global, levando em consideração a luz gerada pela interação entre os objetos. Em um ambiente real, mesmo superfícies não iluminadas diretamente possuem certa cor, pois são

atingidas pela iluminação gerada a partir do reflexo de outros objetos. No modelo de Phong, esta iluminação é simplificada, pois, calcular estas interações é algo extremamente custoso para uma placa gráfica, tornando-se inviável sua utilização para programas que necessitem de uma renderização rápida. A simplificação proposta por Phong é a utilização de uma componente constante de iluminação ambiente, a qual multiplicada pelo coeficiente de reflectividade da superfície e pela sua intensidade de cor difusa resulta na componente de luz ambiente do objeto.

$$I_{amb} = I_A \cdot k_A \cdot C_D \qquad 5.1$$

Onde:

I_{amb}: Intensidade da luz ambiente resultante;

I_A: Intensidade da luz ambiente incidente;

k_A: Coeficiente de reflexão da luz ambiente;

C_D: Intensidade da cor difusa do objeto;

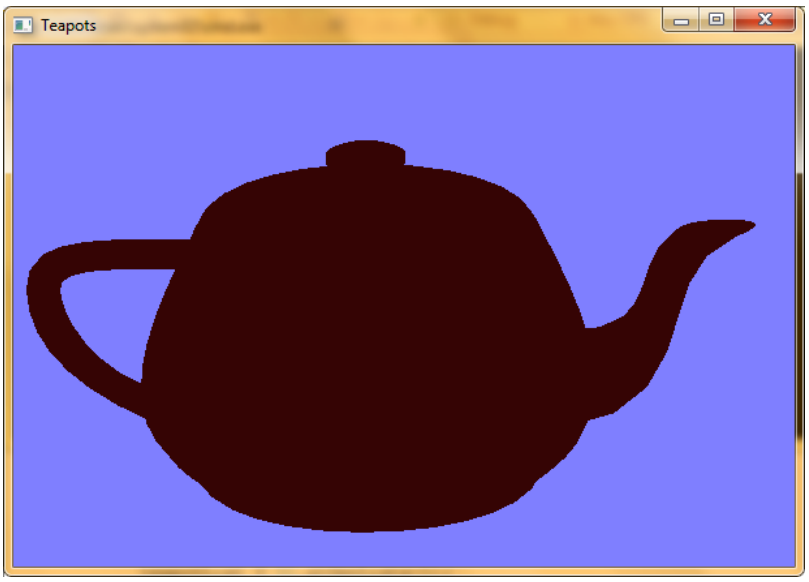

Figura 5.8 - Exemplo de iluminação - Componente ambiente

- Difusa: A componente difusa representa reflexões que não são direcionais, isto é, não depende da posição do observador, pois o objeto emite luz igualmente em todas as direções. Cada objeto possui um coeficiente de refletância difusa para determinar quanta luz é refletida, e a intensidade da reflexão irá variar somente com o cosseno do ângulo entre a direção de incidência do raio de luz com a normal da superfície.

$$I_{dif} = I \cdot k_D \cdot C_D \cdot \cos\theta \qquad 5.2$$

Onde:

I_{dif}: Intensidade da luz difusa resultante;

I: Intensidade da fonte luminosa incidente;

k_D: Coeficiente de reflexão da luz difusa;

C_D: Intensidade da cor difusa do objeto;

θ: Ângulo entre a normal e a direção de incidência da luz;

O cálculo apresentado acima deve ser realizado individualmente para cada fonte luminosa, uma vez que podem ter intensidades e direções de incidência diferentes.

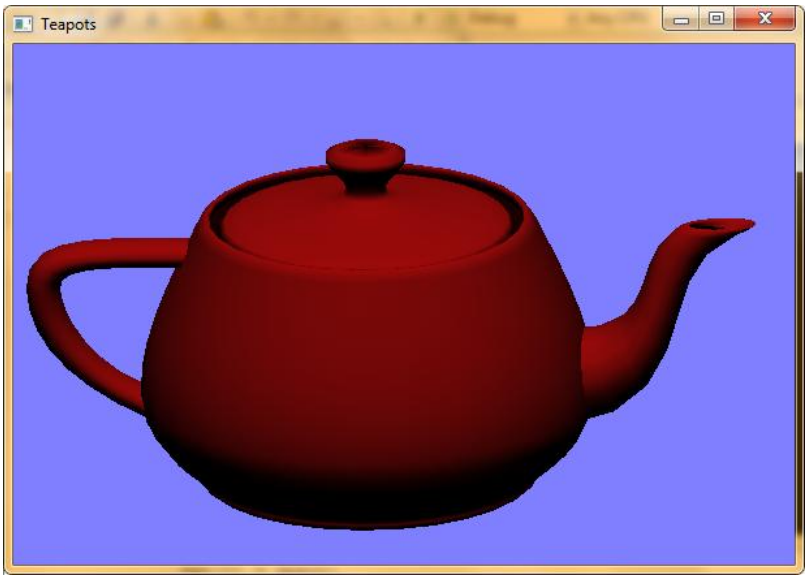

Figura 5.9 - Exemplo de iluminação - Componente difusa

- Especular: A componente especular é utilizada para simular reflexos direcionais, como ocorre em superfícies polidas e/ou brilhantes, onde não ocorre alteração da cor do objeto. A intensidade deste tipo de reflexão é proporcional ao cosseno do ângulo entre a direção da posição do observador e do raio de luz refletido. Utiliza-se ainda um expoente especular, responsável por determinar o quão rápido um reflexo especular decai conforme o ângulo entre a observação e a reflexão aumenta, permitindo simular com perfeição superfícies mais ou menos brilhantes.

$$I_{esp} = I \cdot k_E \cdot C_E \cdot \cos^{n} \beta \qquad 5.3$$

Onde:

I_{esp}: Intensidade da luz especular resultante;

I: Intensidade da fonte luminosa incidente;

k_E: Coeficiente de reflexão da luz especular;

C_E: Intensidade da cor especular do objeto;

n: Expoente especular, responsável por tratar a imperfeição da especularidade.

β: Ângulo entre o raio refletido e o observador;

O cálculo apresentado acima deve ser realizado individualmente para cada fonte luminosa, uma vez que podem ter intensidades e direções de incidência diferentes.

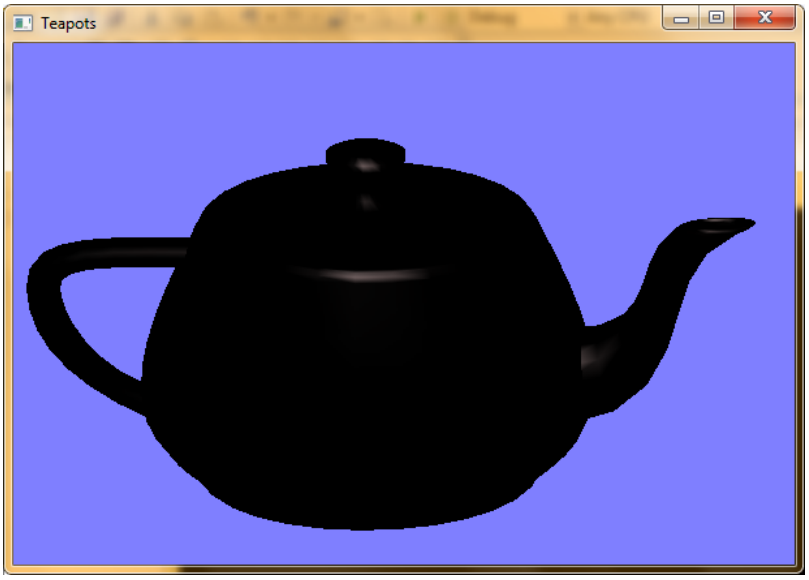

Figura 5.10 - Exemplo de iluminação - Componente especular

A cor final é a soma das três componentes:

$$I = I_{amb} + I_{dif} + I_{esp} \quad 5.4$$

É importante ressaltar que a soma dos coeficientes de reflexão descritos anteriormente deve ser sempre menor ou igual a um quando se tratar de materiais não emissivos, para não haver a sensação de produção de luz a partir do objeto.

$$k_A + k_D + k_E \leq 1 \quad 5.5$$

Como a placas gráficas utilizam o sistema RGB para o cálculo e exibição de cores e os materias possuem coeficientes de reflexão diferentes para cada componente, os cálculos apresentados acima deve ser efetuados separadamente para o vermelho, verde e azul, resultando assim na intensidade final de cada uma das cores separadamente.

5.1.3. Reflexão Global

Existem ainda outras técnicas para o cálculo da iluminação dos objetos que levam em conta a reflexão entre os objetos de uma cena. O método mais conhecido para este cálculo é o Ray Tracing.

Neste método, ocorre a simulação do caminho percorrido por um raio de luz, mesmo após refletir em diversos objetos, porém de trás para frente, ou seja, o raio parte do observador. Isto se deve ao fato de que uma fonte luminosa gera infinitos raios, porém apenas uma pequena parte destes raios realmente atinge o observador e, desta forma, fazendo-se o cálculo a partir da fonte de luz, seria gerada uma grande quantidade de cálculos desnecessários, tornando inviável o processamento pelos computadores existentes hoje em dia.

No algoritmo proposto por este método, tem-se um raio partindo de cada pixel da tela e, após o cálculo de suas reflexões nos objetos da cena, é possível obter a cor do pixel em questão.

A técnica de Ray Tracing é, sem dúvida, muito mais exigente computacionalmente que os algoritmos de reflexão local, porém permite produzir um efeito de foto realismo muito elevado, como pode ser observado na Figura 5.11.

Figura 5.11 - Exemplo de imagem gerada por Ray-Tracing – Extraído de (9)

Devido ao grande processamento necessário para a aplicação desta técnica, ela se torna inviável para aplicações onde a velocidade seja um fator determinante, como jogos, por exemplo, sendo muito utilizado em locais onde é possível um pós processamento da imagem, como em filmes.

5.1.4. Shaders

Como um objeto tridimensional é representado a partir de polígonos em computação gráfica, é necessário um meio de calcular a cor que preenche o polígono, e não somente a cor de seus vértices.

Os shaders são uma parte da pipe-line existente nas placas gráficas responsáveis por estes cálculos, podendo-se aplicar diferentes algoritmos de acordo com o efeito desejado, sendo que o algoritmo é aplicado individualmente para cada pixel da imagem.

5.1.4.1. Flat Shading

O algoritmo de ‘Flat Shading’ é o mais simples de todos e calcula a cor de cada pixel como sendo única em todo o polígono. Desta forma, cada polígono possui apenas uma normal e após o cálculo da cor a mesma é aplicada a todo o polígono.

A Figura 5.12 mostra um exemplo de aplicação desta técnica. Nota-se que as superfícies arredondadas não apresentam um boa aproximação, sendo claramente visível cada polígono, sem a sensação de continuidade e curvatura.

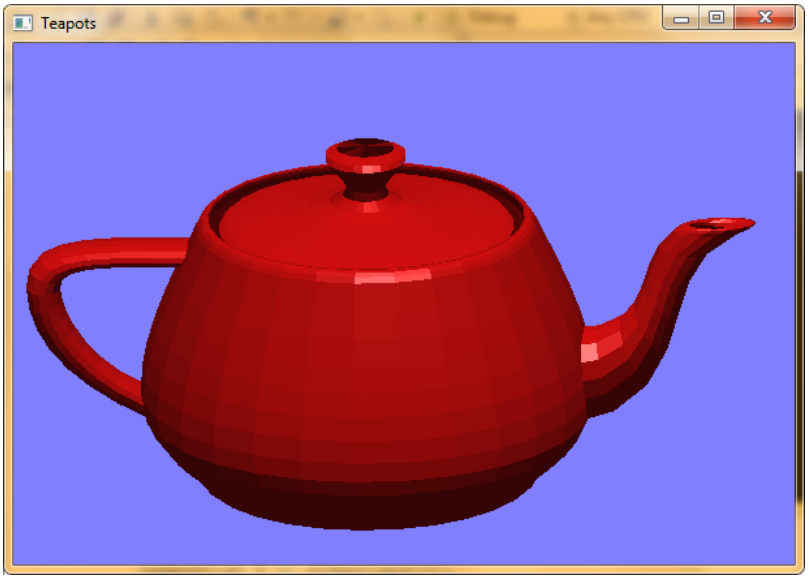

Figura 5.12 - Exemplo de aplicação de Flat Shading

5.1.4.2. Gourraud

O algoritmo de Gourraud é um dos mais utilizados e atualmente está implementado em hardware nas placas gráficas. Neste algoritmo, são calculadas as cores por vértice, e o polígono é preenchido após uma interpolação bi-linear da cor de cada vértice.

A aplicação deste algoritmo de shading permite um grau de aproximação bastante satisfatório, como pode ser observado na Figura 5.13.

É importante observar que, neste caso, a normal de cada vértice é calculada como uma média da normal de cada polígono que compartilha tal vértice, ou definida manualmente.

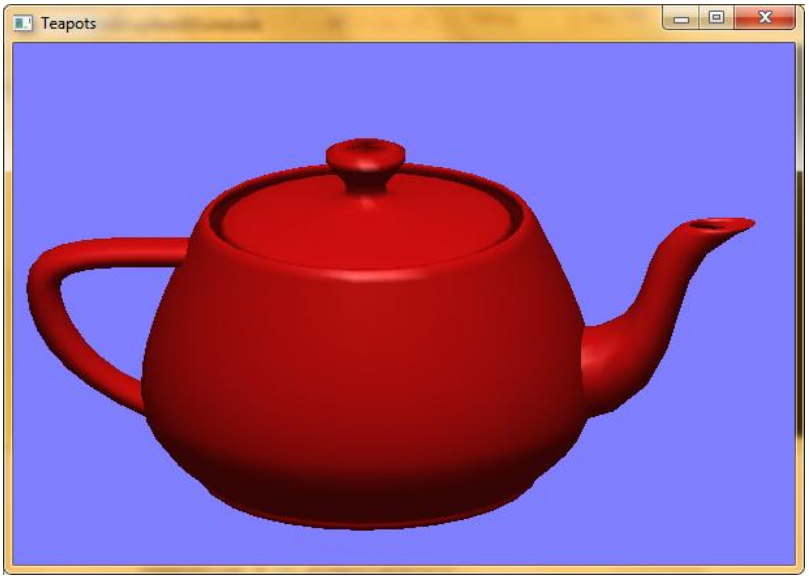

Figura 5.13 - Exemplo de aplicação de shadder – Algoritimo de Gourraud

5.1.4.3. Phong

O algoritmo de Phong, semelhantemente ao Gourraud, faz uma interpolação bi-linear, porém, ao invés de utilizar a cor de cada vértice, utiliza a normal dos vértices e faz um novo cálculo da iluminação resultante no pixel.

Este algoritmo permite que efeitos de reflexão dos objetos especulares sejam reproduzidos mais fielmente, porém produz um aumento substancial no processamento necessário.

5.2. Texturas

Como pôde ser observado no tópico anterior, o cálculo da iluminação real de um objeto em uma cena tridimensional pode ser muito custoso computacionalmente. A aplicação de texturas permite que sejam simulados efeitos de forma a fazer a cena ficar bastante próxima da realidade e ao mesmo tempo, aumentar o desempenho das placas gráficas, pois pode-se diminuir substancialmente a quantidade de cálculos necessários.

Texturas são amplamente utilizadas para a simulação de superfícies complexas de serem modeladas, como madeira e granito, entre outras, para simulação de efeitos de iluminação como sombras e reflexões, além de permitir mapeamento de normais e deslocamentos para os vértices de um modelo tridimensional.

O mapeamento mais utilizado é para simulação de superfícies complexas, onde tem-se uma imagem bidimensional mapeada na superfície tridimensional. Para sua aplicação, deve-se indicar para cada vértice, uma coordenada (x, y) da textura, de forma que ao longo do polígono será feita uma interpolação das coordenadas.

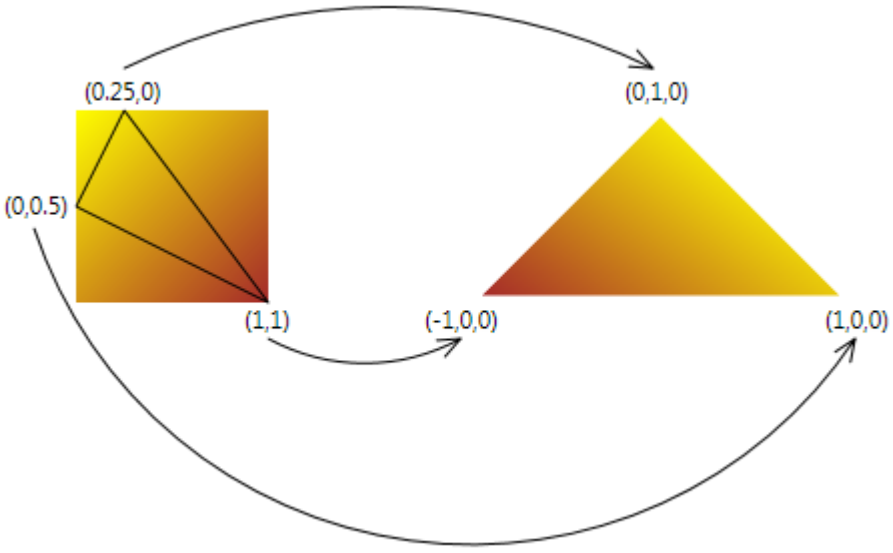

Figura 5.14 - Coordenadas no mapeamento de texturas – Extraído de (3)

Este tipo de mapeamento permite que sejam simulados diversos tipos de materiais, como a madeira, apresentado na Figura 5.15. Pode-se ainda aplicar diversas texturas em um mesmo objeto tridimensional, desde que possuam certa transparência, caso contrário uma iria se sobrepor completamente a outra.

Outra utilização bastante comum desta técnica é a simulação de sombras e reflexões em espelhos. Para este fim, é gerada uma textura dinamicamente, posicionando-se a câmera

na posição desejada, gerando-se uma imagem. Posteriormente, com a câmera em sua posição original, a imagem gerada é mapeada no objeto tridimensional.

Figura 5.15 - Cubo mapeado com textura – Adaptado de (10)

Pode-se utilizar uma textura também como mapa de deslocamentos, e a esta técnica, dá-se o nome de '*Displacement Map*'. Neste caso, é necessário uma imagem em escala de cinzas que representa a variação de altura de um determinado ponto, e por conta disto, deve-se ter uma textura com a mesma quantidade de pontos que a malha do objeto tridimensional. A aplicação deste mapeamento consiste em realizar um deslocamento proporcional à intensidade da cor na textura e é muito utilizado na geração de terrenos com muitas oscilações.

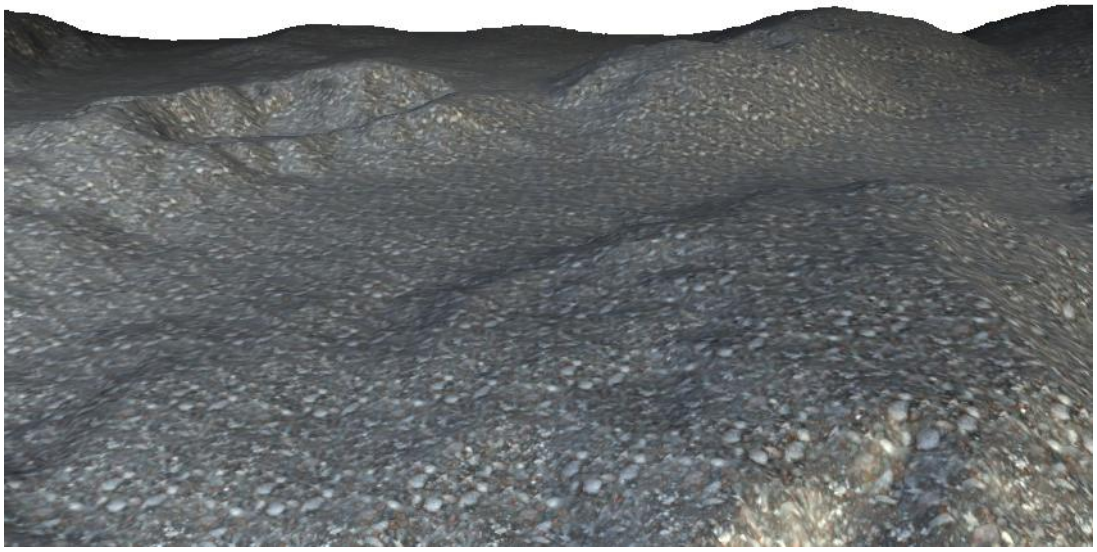

Figura 5.16 - Exemplo de terreno gerado através de mapeamento de deslocamentos – Adaptado de (11)

O ‘*Bump Map*’ é muito semelhante ao ‘*Displacement Map*’, no entanto, a textura representa uma perturbação na normal da superfície ao invés de um deslocamento do vértice. Com a alteração da normal, o cálculo de iluminação irá resultar em diferentes tonalidades, permitindo simular efeitos de rugosidades, por exemplo, sem a necessidade de fazer alterações na malha tridimensional.

Uma vantagem deste tipo de mapeamento em relação ao mapa de deslocamentos é que não é necessária uma malha com mesmas dimensões que a textura, pois as normais são calculadas pixel a pixel durante sua renderização, porém não permite uma grande variação de deslocamentos, pois não ocorrem alterações na malha tridimensional propriamente dita, apenas no cálculo de sua iluminação.

A figura a seguir ilustra um exemplo de uma superfície lisa, seu ‘*bump map*’, e o efeito resultante.

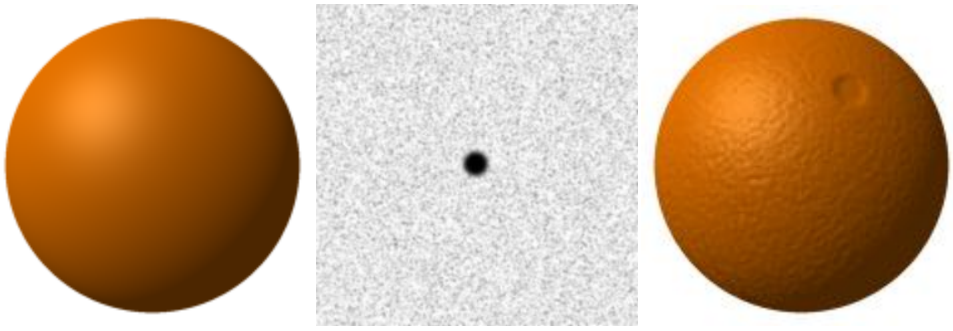

Figura 5.17 - Exemplo de aplicação de 'Bump Map' – Extraído de (12)

Como último modelo de mapeamento, tem-se o mapeamento de normais (*normal map*). No '*bump map*' é feito um mapeamento apenas da intensidade, e desta forma, a textura é composta de uma imagem em escala de cinzas. No mapeamento de normais, a textura é composta de uma imagem RGB, onde cada combinação de vermelho, verde e azul corresponde, respectivamente, às coordenadas X, Y e Z da normal.

Figura 5.18 - Exemplo de normal map – Extraído de (13)

Capítulo 6. Realidade Virtual

O fenômeno de percepção de profundidade, dimensões e distâncias entre objetos é conhecido por estereoscopia, e este ocorre pelo fato do olho esquerdo ter um ponto de observação ligeiramente diferente do olho direito de forma que o cérebro, ao interpretar ambas as imagens consegue produzir tais efeitos.

A realidade virtual consiste em simular a percepção real de uma cena utilizando estereoscopia, de modo que o observador tenha a sensação de imersão tridimensional, mesmo observando uma imagem plana, como por exemplo, o monitor de um computador. Realidade virtual pode englobar ainda a interação entre o usuário e a cena tridimensional, porém estes tópicos fogem do escopo e não serão discutidos neste trabalho.

Pode-se aplicar a realidade virtual simulando duas visões distintas da cena, uma responsável pela visão do olho esquerdo e outra pelo direito.

Foi discutido até agora a visualização por um único ponto de observação (*Mono Eye*), conforme Figura 6.1. Quando se deseja simular o fenômeno de estereoscopia, são necessários dois pontos de observação, um responsável pela visualização do olho esquerdo e outro pela visualização do olho direito. Desta forma, são projetadas duas imagens diferentes, apresentadas independentemente na tela. A Figura 6.2 ilustra esta situação.

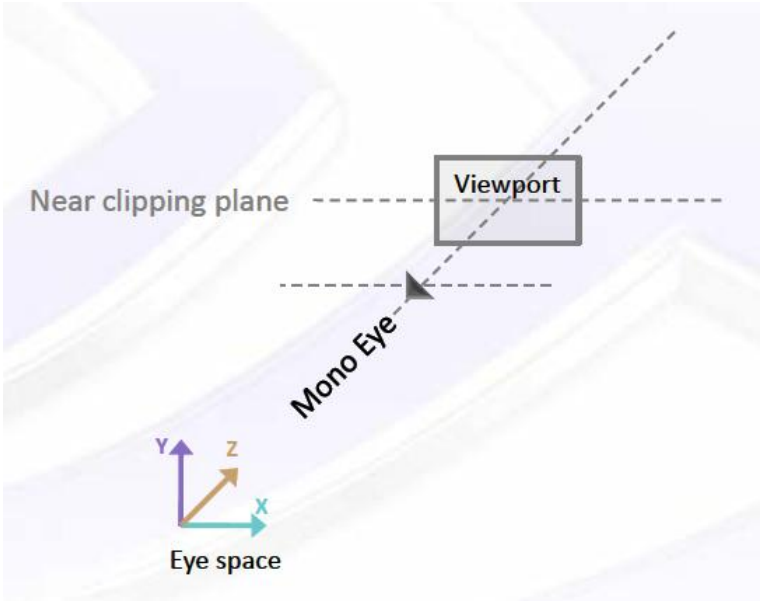

Figura 6.1 - Visualização por um ponto de observação único – Extraído de (14)

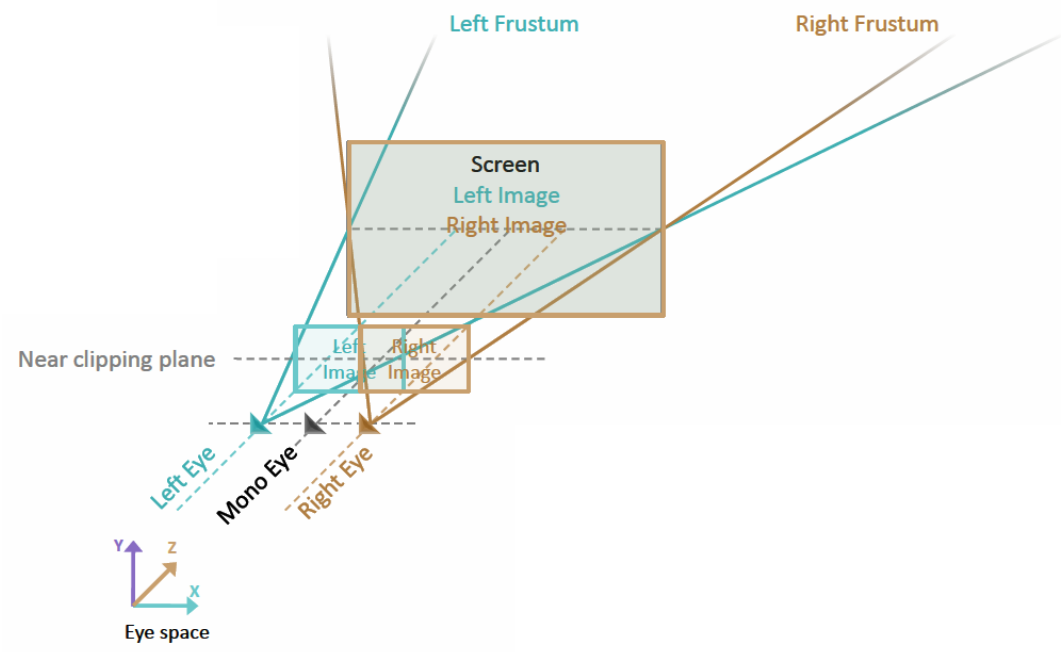

Figura 6.2 - Campo de visão dos olhos esquerdo e direito – Extraído de (14)

Para que uma imagem seja corretamente exibida e se tenha a percepção tridimensional desejada, existem alguns fundamentos de extrema importância:

- Distância inter-axial (Interaxial)
- Profundidade da tela (Screen Depth)
- Paralaxe (Parallax)

A distancia inter-axial está relacionada com a distância entre os dois pontos de visualização, ou seja, a distância entre os pontos de observação do olho esquerdo e do direito, e deve ser próxima da distância real entre os olhos de uma pessoa (Figura 6.3).

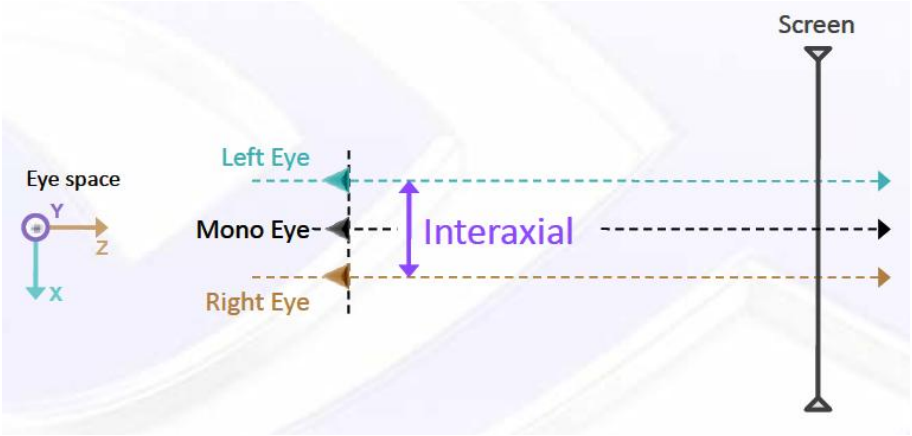

Figura 6.3 - Representação da distância inter-axial – Extraído de (14)

A profundidade da tela, também chamada de convergência, está relacionada com a distância virtual da tela, e é representada pelo plano onde os campos de visão do olho esquerdo e direito se interceptam (Figura 6.4).

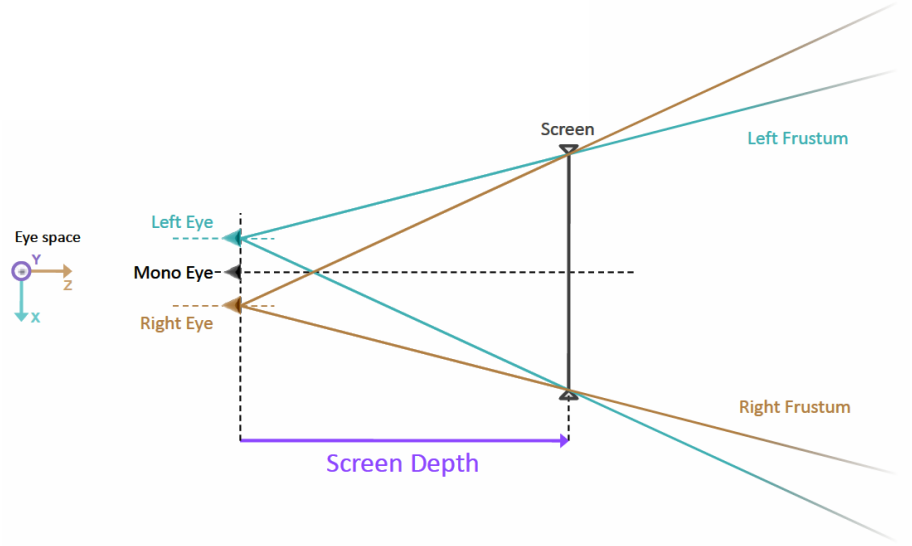

Figura 6.4 - Representação da profundidade da tela – Extraído de (14)

Com esta definição, fica fácil perceber que as matrizes de visualização de cada olho diferem apenas no fato de transladarem a coordenada X para a esquerda ou direita.

Por último, tem-se a paralaxe. Ao se traçar uma reta entre o ponto de visualização do olho esquerdo e o ponto do objeto tridimensional e outra reta entre o ponto de visualização do olho direito e o mesmo ponto do objeto, estas interceptam o plano da tela em diferentes posições. O nome dado à distância entre estes dois pontos projetados é paralaxe.

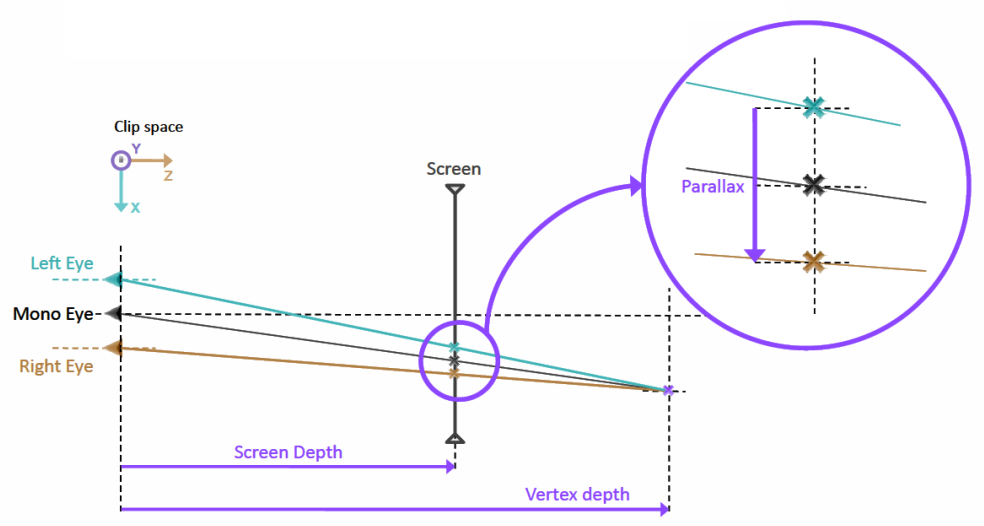

Figura 6.5 - Exemplo de paralaxe – Extraído de (14)

A paralaxe é uma função da profundidade do ponto (*W*), máxima paralaxe para pontos no infinito (*DepthFactor*) e profundidade da tela (*ScreenDepth*), e pode ser representada pela seguinte equação:

$$Paralaxe = MáximaParalaxe \cdot (1 - ProfundidadeDaTela/W) \quad 6.1$$

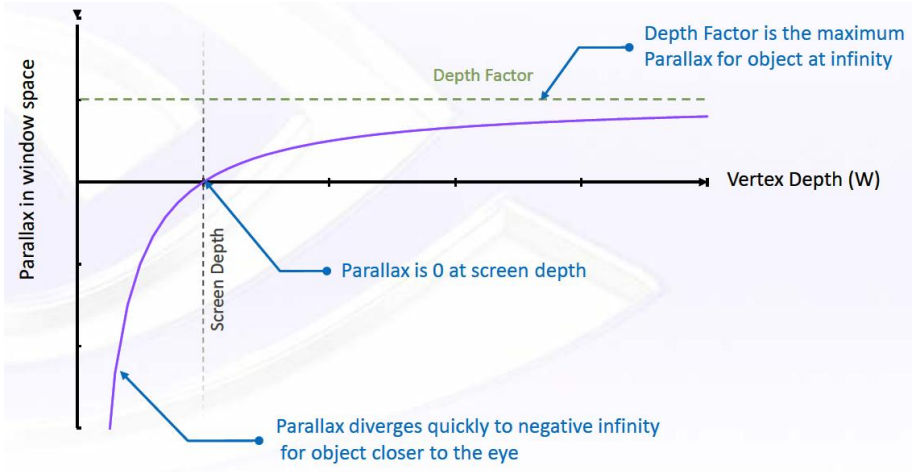

Figura 6.6 - Variação da paralaxe – Extraído de (14)

Apesar da teoria ser algo relativamente simples, a maior dificuldade de aplicação desta técnica está em como conseguir fazer a apresentação de ambas as imagens independentemente e simultaneamente, de forma que o cérebro consiga fazer a reconstituição da cena tridimensional real. Para tal, foram desenvolvidas técnicas, sendo as principais apresentadas a seguir.

6.1. Por Cores

Este modelo consiste em colorir as imagens com cores diferentes para o olho esquerdo e o olho direito. Normalmente, utiliza-se as cores vermelho para o olho esquerdo e ciano para o direito. Tendo-se ambas as imagens coloridas com tons diferentes, pode-se exibi-las simultaneamente na tela. O observador deve, no entanto, utilizar um óculos com filtros vermelho e ciano, de forma que apenas a imagem vermelha chegue ao olho esquerdo e apenas a ciano chegue ao olho direito.

Os óculos podem ser feitos facilmente com papéis coloridos. Um ponto forte desta técnica é o fato de ser extremamente barata. Um ponto fracoé a restrição ao uso de cores no desenho a ser exibido. A Figura 6.7 ilustra um exemplo de imagem produzida para visualização por cores.

Figura 6.7 - Exemplo de estereoscopia por cores – Extraído de (15)

6.2. Por Frequência

A utilização de frequência requer equipamentos específicos e consiste em apresentar as imagens intercaladas a cada atualização de tela. Para que esta atualização seja imperceptível ao observador, é necessária a utilização de frequências elevadas, acima de 120Hz. Como cada imagem irá aparecer em apenas metade das telas exibidas, cada olho perceberá uma taxa de atualização de 60Hz, para uma frequência efetiva de 120Hz.

O observador deverá ainda utilizar um óculos que, sincronizado com o monitor, irá ocultar a visão de cada olho de acordo com a imagem exibida na tela, de forma que cada olho enxergue apenas a imagem correspondente ao seu ponto de vista.

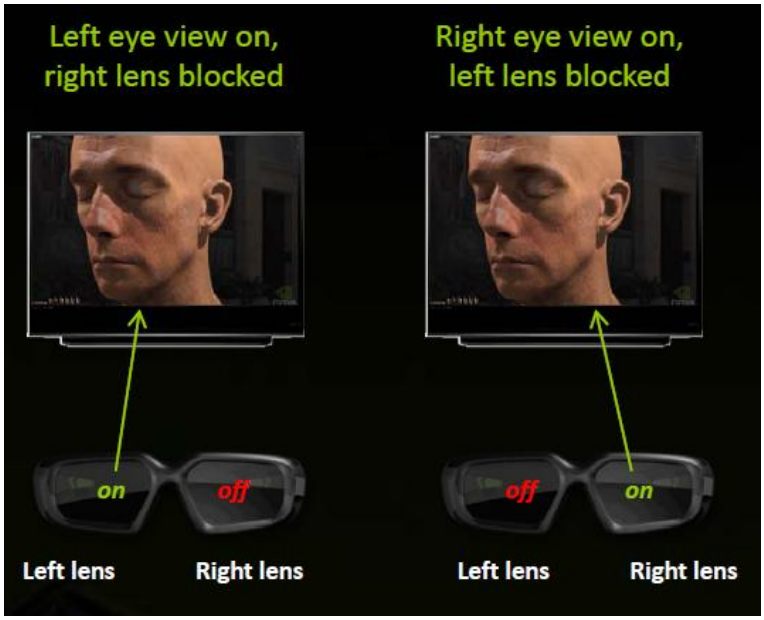

Figura 6.8 - Funcionamento da estereoscopia por frequência – Extraído de (14)

6.3.Por Polarização

Esta é a técnica mais utilizada em cinemas 3D, e consiste em polarizar os raios luminosos em determinadas direções.

Uma onda luminosa comum pode possuir qualquer direção de oscilação, no entanto, é possível a polarização desta direção, de forma a fazê-la se propagar apenas na direção desejada. Com isto, pode-se fazer com que as imagens referentes aos dois olhos sejam polarizadas em diferentes direções.

Para o espectador conseguir enxergar a cena de forma desejada, este deverá utilizar um óculos que possua filtros polarizados em direções diferentes para o olho esquerdo e olho direito, e desta forma, ao observar a cena, cada olho será capaz de enxergar apenas a imagem polarizada na mesma direção que a lente, permitindo ao cérebro fazer a fusão das duas imagens e gerar o efeito desejado.

Capítulo 7. Requisitos e Bibliotecas Gráficas

A biblioteca utilizada para a visualização tridimensional é de extrema importância, pois será determinante para a implementação dos recursos desejados, além de ser fundamental para a criação de um software com o melhor desempenho possível. É também necessário que ela possua suporte a todos os requisitos gráficos do software que será desenvolvido.

Desta forma, foram primeiramente levantados os principais requisitos gráficos que o software deverá cumprir e, posteriormente, visando as necessidades do projeto, estudaram-se três biblioteacas para visualização tridimensional: *OpenGL*, *WPF* e *XNA*.

7.1. Requisitos Gráficos

Tendo em vista o objetivo do software a ser desenvolvido, foram levantados os principais requisitos necessários e desejáveis para os quais a biblioteca gráfica deve ter suporte.

Os principais requisitos são:

1. Integração com C#: Tem-se a restrição de utilização do C# como linguagem de programação para o desenvolvimento do software, uma vez que foi a linguagem utilizada no módulo de projeto e visualização bidimensional. Desta forma, a biblioteca gráfica deve possuir um meio de ser integrada ao C#.

2. Alto desempenho: A aplicação deverá utilizar os recursos gráficos da placa de vídeo para obter um maior desempenho, uma vez que estarão presentes formas complexas (com elevado número de vértices), além dos efeitos de iluminação e textura. Um processamento apenas por software tornaria a utilização da aplicação inviável.

3. Permitir a utilização de diferentes tipos de materiais (especulares, emissivos, etc) e aplicação de texturas: deve ser possível utilizar diferentes tipos de materiais e aplicação de texturas para que o cabo projetado seja representado da forma mais realista possível.

4. Possibilidade de alterar a cor de cada vértice separadamente: é necessário que seja possível a alteração da cor de cada vértice separadamente para a representação dos resultados de uma eventual simulação.

5. Permitir a criação de animações: deve ser possível criar animações para visualizar deslocamentos calculados em uma eventual simulação, de forma a proporcionar uma perspectiva real do comportamento do cabo sendo projetado. A animação deve, no entanto, ser implementada sem que exista perda de desempenho do software.

6. Permitir criação de estereoscopia por cores: é necessário ainda que possa ser criado o efeito de realidade virtual utilizando cores (anaglyphs) para trazer ao usuário uma perspectiva mais real do cabo projetado.

Outros itens desejáveis, porém não imprescindíveis são:

7. Fácil implementação e boa documentação da biblioteca: devido ao tempo disponível para o desenvolvimento e o porte do software previsto, é desejável que se tenha uma biblioteca com uma implementação razoavelmente fácil e intuitiva, com vasta gama de referências para que seja possível a implementação de todos os requisitos.

8. Possibilidade de customização da iluminação: é desejável que se possa fazer uso de diferentes tipos de fontes luminosas, além da aplicação de *shaders*, permitindo um nível de realidade superior e um bom desempenho da aplicação.

9. Possibilidade de implementação de estereoscopia por frequência: apesar da estereoscopia por cores ser um requisito, a mesma possui uma certa restrição às cores, pois as utiliza como filtros, restringindo as cores possíveis de serem visualizadas. Desta forma, seria desejável que a biblioteca permitisse implementação de estereoscopia por freqüência, pois

manteria total fidelidade às cores da cena e causaria uma melhor experiência de realidade virtual ao usuário.

7.2. OpenGL

O *OpenGL* é uma ferramenta de alta performance para gerar desenhos tanto bidimensionais quanto tridimensionais, pertencente a SGI (*Silicon Graphics International*) e de código aberto.

Esta biblioteca possui implementação em C++, utilizando assim código não gerenciado. No entanto, é possível sua utilização com C# fazendo-se a importação das DLLs (acrônimo para *Dynamic Link Library* do inglês), mas devido ao grande número de funções existentes no *OpenGL*, este trabalho seria inviável de ser feito dentro do prazo determinado, tornando-se necessária a utilização de frameworks com esta implementação feita previamente, como por exemplo o *TAO Framework*.

Devido ao fato desta biblioteca ser feita nativamente em uma linguagem de mais baixo nível, ela possui acesso direto aos recursos gráficos de hardware (da placa de vídeo), tendo um desempenho bastante elevado. Por outro lado, a mesma possui uma implementação mais difícil quando comparada a soluções feitas com código gerenciado. No entanto, devido a sua grande popularidade e por já estar consolidada no mercado, possui uma documentação bastante completa incluindo diversos sites e livros sobre o assunto.

O *OpenGL* também possui recursos bastante completos no que diz respeito a iluminação, materiais e texturas. Utilizando-se esta biblioteca, é possível a definição de todos os tipos de fontes de luz vistas anteriormente (ambiente, direcional, pontual, spot-light), além da definição de diferentes intensidades e cores para cada componente utilizada pelo algoritmo de *Phong* (ambiente, difusa e especular). Pode-se ainda definir materiais diferentes para a face

frontal e posterior de cada polígono e, de maneira semelhante às fontes luminosas, é possível diferenciar as cores para os cálculos de reflexão ambiente, difusa e especular. Adicionalmente, é possível a criação de um material emissivo, ou seja, que possui luz própria.

Existe também a possibilidade de utilização de *shaders* específicos, produzidos em linguagem própria (*GLSL*, *OpenGL Shading Language*), o que permite uma grande customização e implementação de diversos efeitos diferenciados, além de possuir os algoritmos de *Flat Shading* e de *Gourraud* já implementados.

O *OpenGL* permite o controle de cores por vértices, facilitando assim a implementação da visualização de resultados da simulação do cabo. No entanto, apesar de ser uma biblioteca direcionada para jogos, que requerem a utilização de animações em grande escala, estas são feitas normalmente utilizando-se matrizes de transformação e não possui recursos nativos para a aplicação da animação necessária neste projeto, onde se tem diferentes deslocamentos para cada vértice separadamente sendo, portanto, necessário o cálculo manual dos vértices em cada momento da animação.

O software em desenvolvimento prevê também a implementação de realidade virtual (estereoscopia) para o modelo 3D. Desta forma, é fundamental que o módulo tridimensional seja criado em uma plataforma que suporte este tipo de aplicação. Fazendo utilização de transparências e sobreposição de duas imagens, vistas de pontos diferentes, é possível a criação de estereoscopia por cores. Além disto, o *OpenGL* também possui uma biblioteca de alto desempenho para realização de estereoscopia por frequência, conhecida por *OpenGL Performer*™ e comercializada pela SGI.

OpenGL Performer™ é um conjunto de ferramentas direcionadas para a implementação de aplicativos de realidade virtual, com bibliotecas para desenvolvimento nas linguagens C e C++, sendo que no caso de sua utilização em C#, seria necessária a importação

das funções a partir da biblioteca de vínculo dinâmico (*DLL*) para serem utilizadas em C#. Esta ferramenta é compatível com os sistemas operacionais *IRIX*, *Linux* e *Windows* (16).

7.3. WPF 3D

Microsoft *Windows Presentation Fundation* (*WPF*) foi inicialmente desenvolvido para programação de interface de aplicativos para o *Windows Vista* e *Windows 7*, porém também pode ser executado em versões anteriores do Windows, uma vez que o *WPF* é parte integrante do *.NET Framework 3.0* e posteriores, sendo necessário, apenas, a instalação deste pacote (17).

Esta ferramenta possui duas principais interfaces de programação relacionadas, sendo possível escrever programas totalmente em C#, ou uma nova linguagem baseada em XML chamada *Extensible Application Markup Language* (*XMAL*, pronunciada ‘zammel’), podendo ainda ser utilizada as duas formas de programação os mesmo tempo, fazendo parte do código em C# e parte em *XMAL* (17).

O *WPF* é direcionado para a criação de interface de aplicativos, porém possui diversas ferramentas para desenhos tridimensionais. Devido ao fato desta ferramenta não fazer utilização dos recursos gráficos da placa de vídeo diretamente e, assim, tendo grande parte de sua implementação feita em software, a mesma não apresenta um desempenho significativo, tornando o processamento de objetos complexos bastante custoso.

Por outro lado, devido ao fato desta ferramenta ser parte integrante do *.NET Framework*, a mesma possui um implementação bastante facilitada, além de uma documentação completa disponibilizada pela *Microsoft* e alguns livros.

No *WPF*, os materiais são aplicados ao objeto como um todo, permitindo apenas a definição de materiais diferentes para os lados frontais e posteriores. É possível a definição de

materiais difusos, especulares e emissivos, sendo que no caso da utilização de iluminação ambiente, a cor é calculada baseada na componente difusa do objeto.

Existe ainda a implementação de todos os tipos de fontes luminosas discutidas anteriormente. No entanto, diferentemente do *OpenGL*, cada fonte luminosa possui apenas uma cor, sem distinção entre componentes ambiente, difusa e especular. Devido ao fato do *WPF* não possuir acesso à placa gráfica, outras customizações de cores e iluminações, como a implementação de *shaders*, não são possíveis com esta ferramenta.

Como os materiais são tratados em nível de objeto ao invés de vértice, não é possível a determinação da cor de cada vértice separadamente. Para contornar esta restrição, é possível a utilização de texturas e, ao criar uma textura com as cores desejadas, basta mapear um ponto da textura com o vértice correspondente.

Por outro lado, o *WPF* possui implementações prontas para a aplicação de animações, da forma necessária para a aplicação a ser desenvolvida, permitindo a animação de vértices independentemente, sem a necessidade de calcular manualmente todas as posições.

É possível ainda realizar no *WPF* modelos de realidade virtual utilizando polarização de imagens por cor. A Figura 7.1 mostra um exemplo de aplicação desta técnica em *WPF*. Por outro lado, não foi encontrada documentação para implementação de realidade virtual por freqüência utilizando o *WPF*, assumindo-se portanto, para efeitos de decisão neste projeto, que não existe suporte para tal na biblioteca em questão.

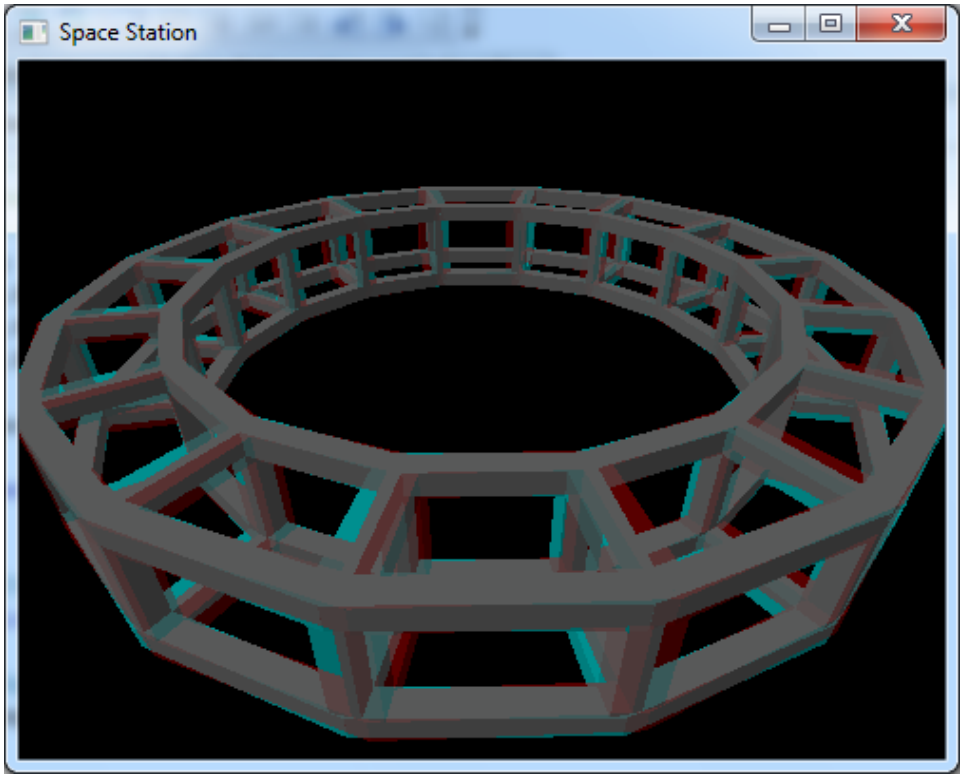

Figura 7.1 - Estereoscopia por cores utilizando WPF – Extraído de (3)

7.4.XNA

O *XNA* é um framework integrante do pacote *XNA Game Studio*, distribuído gratuitamente pela *Microsoft*, sendo uma ferramenta gráfica direcionada para o desenvolvimento de jogos para *Windows* e *Xbox 360*. Anunciada pela primeira vez em 2006, está, atualmente, em sua versão 4.0, que inclui suporte para *Windows Phone 7*.

Este pacote gráfico vem a ser um substituto do *Managed DirectX*, ou seja, é uma versão baseada em funcionalidades do *DirectX*, porém direcionada para código gerenciado, sendo uma ótima alternativa para utilização junto ao C#. Devido ao fato de ser baseado em *DirectX*, o *XNA* é uma opção de alto desempenho, com implementação facilitada por ser implementada em código gerenciado.

Tem-se, no entanto, a desvantagem de sua documentação ainda não se encontrar completa, pelo fato de ser uma ferramenta nova, tendo-se poucas referências, apenas fóruns de internet, que funcionam como fonte de informação não oficial.

Devido a fatores de compatibilidade com o *Xbox 360*, o *XNA* não faz uso da pipeline fixa das placas gráficas, implementando, portanto, todos os recursos de iluminação e renderização por *shaders*. No entanto, possui uma classe que faz esta implementação de forma simplificada, simulando alguns dos recursos básicos, como a iluminação ambiente e direcional, além dos materiais difusos, emissivos e especulares. Outra possibilidade que esta classe permite é a coloração por vértices isoladamente.

Caso se deseje a utilização de outras fontes luminosas, como a luz pontual ou o *spot-light*, deve ser criado um *shader* específico. Por outro lado, a implementação utilizando-se *shaders* torna a ferramenta totalmente flexível, permitindo a criação de diversos efeitos diferentes, e uma vez que são executados pelo processador gráfico, mantém-se um alto desempenho.

O XNA, igualmente ao *OpenGL*, não possui suporte direto para a animação da forma que é necessária para projeto, porém, possui a possibilidade de implementação manual da posição dos vértices em cada etapa da animação. Permite, ainda, a utilização de realidade virtual por cores, fazendo uso de transparências e sobreposição de imagens. Possui, também, suporte direto para implementação de realidade virtual por frequência.

7.5. Comparação e Escolha da Biblioteca Gráfica

Após analise dos recursos disponibilizados por cada biblioteca gráfica estudada, foi possível fazer uma comparação entre quais requisitos são atendidos por cada uma, conforme a tabela abaixo.

Tabela 7.1 - Comparação entre bibliotecas gráficas

	Requisitos	1	2	3	4	5	6	7	8	9
Bibliotecas	OpenGL	○(1)	●	●	●	○(2)	●	○(3)	●	●
	WPF 3D	●		●	○(4)	●	●	●		
	XNA	●	●	●	●	○(2)	●	○(5)	●	●

● – Cumpre totalmente o requisito

○ – Cumpre parcialmente o requisito ou um trabalho adicional é necessário.

(1) A integração com C# é possível, porém existe a necessidade de importação de bibliotecas.

(2) A animação deve ser implementada manualmente, não possuindo recursos nativos para tal.

(3) Existe uma boa documentação, porém a implementação requer maior cuidado e é mais trabalhosa.

(4) A definição de cor de cada vértice não é suportada, devendo ser utilizada uma textura para simular tal efeito.

(5) O *XNA* é uma ferramenta de fácil implementação, porém possui pouca documentação e referências sobre o assunto.

Observando-se a tabela, pode-se verificar que o *WPF* não cumpre um requisito essencial referente ao desempenho, além de exigir um trabalho adicional para a apresentação de resultados de simulação, que requerem a definição de cores por vértices, e não possuir suporte para outros requisitos desejáveis. O *WPF* se demonstra uma ferramenta inviável para a utilização neste projeto.

O *OpenGL* e o *XNA*, ambos cumprem todos os requisitos básicos, apesar da necessidade de implementação manual de animações, porém, como o *OpenGL* não possui suporte direto para utilização junto ao C#, sendo necessária a importação das funções, além de possuir um implementação dificultada quando comparada ao *XNA*, este projeto optará pela utilização do *XNA* como biblioteca gráfica, pois, dentre as opções estudadas, é a que melhor se encaixa aos requisitos do projeto

Capítulo 8. Especificações do Módulo Tridimensional

Com base em todo o material estudado até o presente momento, foi dado início à implementação do módulo tridimensional propriamente dito. Para isso, foram levantados os requisitos iniciais da ferramenta, divididos em ‘Requisitos Funcionais’, ‘Requisitos Não Funcionais’ e ‘Requisitos de Interface’, como apresentado a seguir:

Como requisitos funcionais, têm-se:

- Permitir funções de Zoom, Pan e Rotate.
- Possibilitar a visualização do cabo projetado como sólido, estrutura de arames, ou ambas.
- Permitir carregar resultados da simulação de um componente.
- Exibir as deformações com escala de cores correspondente.
- Permitir a animação dos resultados, com opções de Play, Pause e Stop.
- Criação de efeitos de estereoscopia para o cabo projetado
- Permitir a visualização com efeitos de iluminação
- Implementação de textura para os componentes

Como requisitos não funcionais têm-se:

- Facilidade de uso: poucas horas de treinamento permitem o uso das funções básicas
- Evitar falhas: o software encarrega-se de descrever bem o problema e não executa a ação, evitando que este feche – estabilidade do programa

Como requisitos de interface têm-se:

- Permitir ao usuário visualizar e interagir com o cabo projetado.
- Permitir que barras de ferramentas possam ser ocultadas e reposicionadas.

- Permitir ao usuário visualizar a estrutura do cabo, separando as instâncias em camadas.
- Alterar opções do programa como cores do fundo e refinamento da malha.
- Mensagens de erro claras e elucidativas

Com base nos requisitos listados anteriormente, foi criado um esboço da interface a ser criada para abrigar todas as funcionalidades desejadas, como mostrado na Figura 8.1.

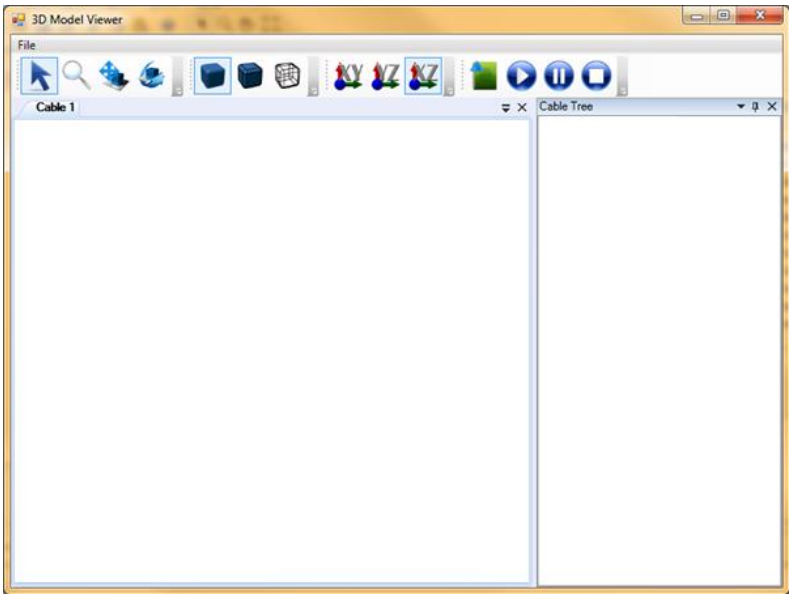

Figura 8.1 - Esboço inicial da interface do módulo tridimensional

Nesta figura, pode-se observar três regiões principais. Na parte superior, está presente uma região para abrigar as barras de ferramentas, as quais darão acesso a todas as funcionalidades do programa. Na região direita, há uma área reservada para a visualização da estrutura do cabo, permitindo ao usuário verificar os componentes presentes. Por último, na região central tem-se uma área reservada para a visualização do cabo gerado. Esta região também será responsável por eventuais interações do usuário com o programa, como seleção de componentes ou operações de visualização, como por exemplo, rotacionar o cabo para visualizar uma região escondida.

Capítulo 9. Projeto do Software

Existem muitas eventualidades que podem ocorrer durante o desenvolvimento de um software: detalhes não previstos no projeto inicial, mudança por parte dos clientes, problemas de implementação e muitos tipos de erros. Quando um bom projeto de software é feito, muitos desses problemas têm impacto reduzido, tanto em termos de tempo gasto quanto em custos de desenvolvimento.

Se há um projeto claro e definido, redundâncias são eliminadas, resultando em códigos enxutos, legíveis e com chances menores de conter erros. Modificações são facilitadas no software, pois estas são feitas no projeto antes de serem efetivamente implementadas. Essas mudanças são mais locais e em pontos determinados.

Para que um software seja bem feito, a sua confecção deve seguir determinadas regras. Existem diversos modelos para o desenvolvimento de um software como, por exemplo, o modelo em cascata, o modelo incremental, o modelo RAD (*Rapid Application Development*), o modelo em espiral ou evolutivo, dentre outros. Para o presente projeto, optou-se por utilizar a metodologia de desenvolvimento em espiral, representada esquematicamente na Figura 9.1.

O modelo em espiral ou evolutivo é um processo iterativo no qual nem todos os requisitos são conhecidos a priori. Em certos tipos de problema, os requisitos surgem conforme as iterações do software são realizadas. Nesses casos, o modelo espiral é o mais indicado, pois o software passa constantemente pelas cinco etapas apresentadas na Figura 9.1, tendo seus requisitos revistos e ajustados conforme a necessidade.

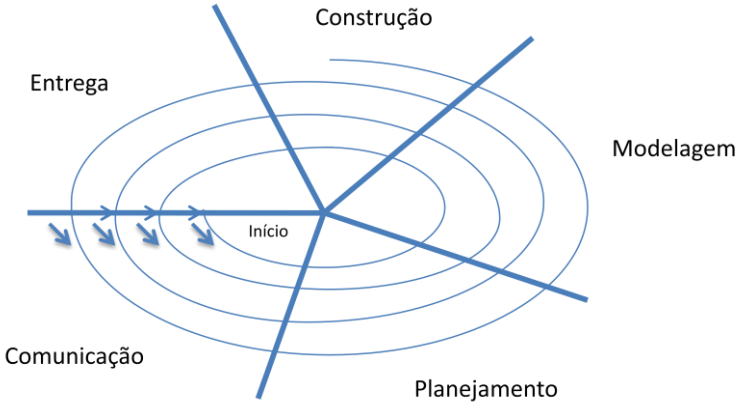

Figura 9.1 - Modelo em Espiral

Definida a metodologia a ser adotada, o planejamento do software foi feito baseado em diagramas de fluxo de dados e de casos de uso, descritos a seguir.

9.1.Diagramas de Fluxo de Dados

Os diagramas de fluxo de dados mostram o fluxo de informação dentro do software. Tais diagramas podem exibir camadas mais profundas do software, sendo que uma das ações do primeiro diagrama é um novo diagrama e assim por diante.

Para o software em questão, foram construídos quatro destes diagramas. O primeiro, exibido na Figura 9.2, mostra a estrutura do software como um todo.

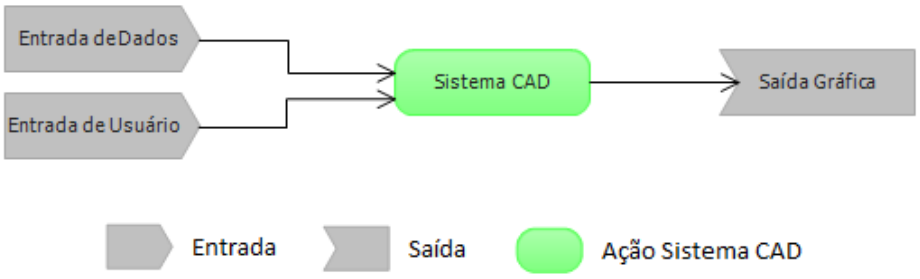

Figura 9.2 - Diagrama de Fluxo de Dados Nível 0

Neste diagrama, tem-se como entradas as ações do usuário, assim como uma entrada de dados para a geração do modelo a ser analisado e, como saída, a visualização gráfica tridimensional, objetivo principal do software a ser desenvolvido. Também se vê a ação

denominada *'Sistema CAD'* que representa o sistema como um todo. Assim, percebe-se como o programa funciona, de uma maneira geral. Essa ação é então aberta em um novo diagrama, exibido na Figura 9.3.

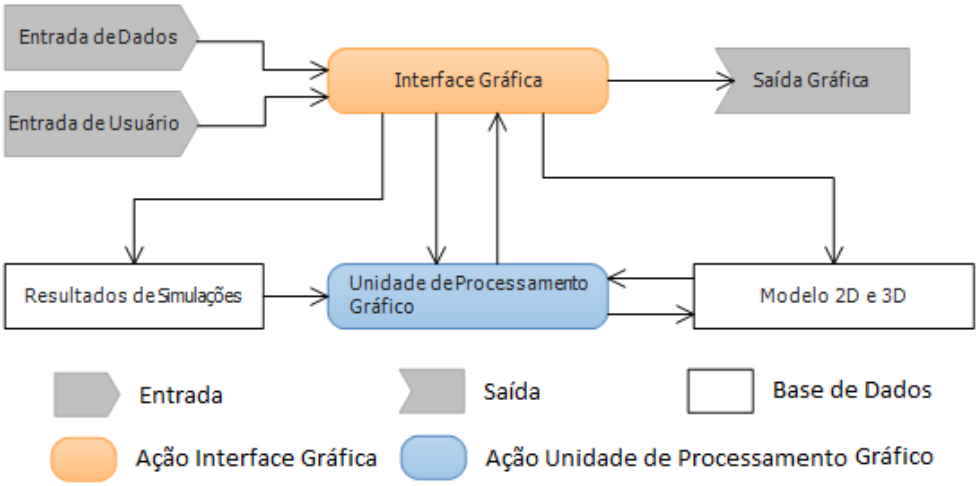

Figura 9.3 - Diagrama de Fluxo de Dados Nível 1

Neste novo diagrama tem-se a ação *'Sistema CAD'* subdividida em duas outras ações denominadas '*Interface Gráfica*', responsável por gerar a interface gráfica para o usuário, e *'Unidade de Processamento Gráfico'*, a qual é responsável por fazer todo o processamento gráfico do modelo tridimensional. Criou-se ainda duas bases de dados principais, uma denominada *'Modelo 2D e 3D'*, a qual armazena todas as informações necessárias, seja tanto do modelo bidimensional, quanto do modelo tridimensional, e uma segunda base de dados (*Resultados de Simulações*) responsável por armazenar as informações dos resultados de uma análise de tensões e/ou deformações para que possa ser feita a visualização no modelo 3D.

Novamente, cria-se um diagrama exibindo uma estrutura mais interna do software, de forma a detalhar a ação denominada *'Unidade de Processamento Gráfico'* como pode ser observado na Figura 9.4.

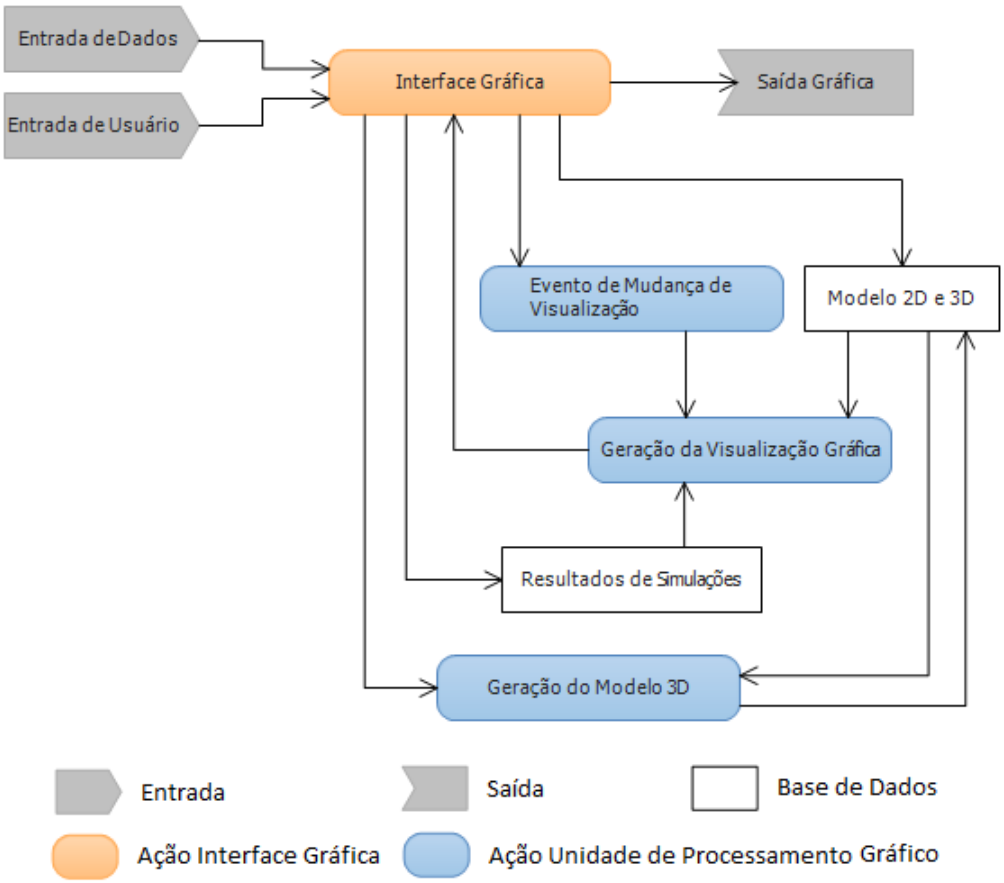

Figura 9.4 - Diagrama de Fluxo de Dados Nível 2 (Unidade de Processamento Gráfico)

Pode se observar que a unidade de processamento gráfico possui basicamente três ações principais que deve ser capaz de executar. Primeiramente, a base de dados do modelo tridimensional deve ser gerada a partir do modelo bidimensional (ação *'Geração do Modelo 3D'*). Tendo esta base de dados gerada, a unidade deve ser capaz de tratar eventos de mudança da visualização atual do modelo, como rotações, arrastos, iniciar modo de realidade virtual, dentre outros, e desta forma, gerar a visualização gráfica correspondente de acordo com o tipo de visualização selecionada.

Tendo a unidade de processamento gráfico devidamente descrita, fez-se um último diagrama de forma a detalhar a ação *'Interface Gráfica'*, como mostrado na Figura 9.5

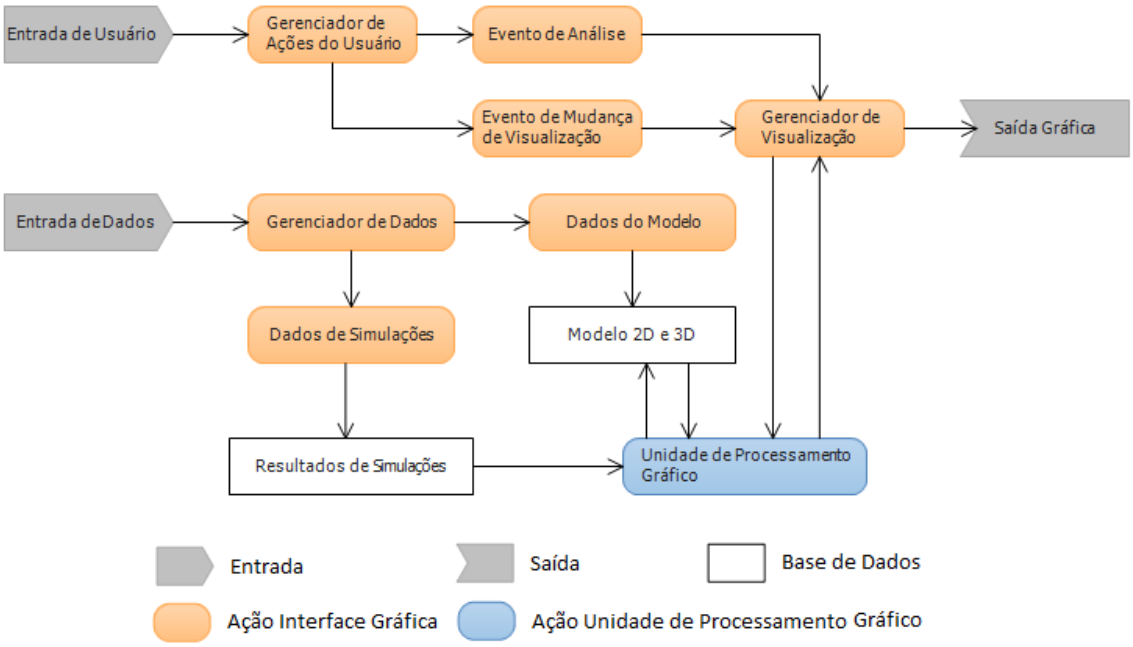

Figura 9.5 - Diagrama de Fluxo de Dados Nível 2 (Interface Gráfica)

Neste último diagrama, a ação *'Interface Gráfica'* é descrita. Inicialmente, a entrada de dados é recebida por um gerenciador (*Gerenciador de Dados*), de forma a direcionar os diferentes tipos de dados que podem existir. No caso de ser recebido um dado de modelo, a ação passa para um gerenciador específico (*Dados do Modelo*), e após tratar o dado de entrada, o mesmo é armazenado na base de dados correspondente Por outro lado, caso seja um dado de resultados de simulações, é enviado para o gerenciador de análises (*Dados de Simulação*), o qual é responsável por criar a base de dados *'Resultados de Simulações'*.

Para o tratamento das ações de usuário, tem-se um gerenciador específico (*Gerenciador de Ações do Usuário*). Para ações referentes a pedidos de análise de simulações e mudanças no tipo de visualização, as ações *'Evento de Análise'* e *'Evento de Mudança de Visualização'* são acionadas, respectivamente. Como todas estas ações requerem algum tipo de alteração na imagem atualmente exibida pelo software, a ação *'Gerenciador de Visualização'* é utilizada, a qual se relaciona com o módulo de processamento gráfico,

enviando as especificações da visualização desejada e ao receber seu retorno, gera a saída desejada (*Saída Gráfica*).

9.2. Casos de Uso

Apenas um mecanismo de descrição do software é, em geral, insuficiente, já que alguns dos detalhes podem ser desconsiderados ou serem despercebidos, dependendo da abordagem escolhida. Para minimizar esses inconvenientes e evitar um posterior retrabalho, mais de uma abordagem foi utilizada. A abordagem anterior, a dos diagramas de fluxos de dados serve para ter uma visão completa do software, principalmente da maneira que este se comporta dadas as entradas do usuário. Os casos de uso são outra abordagem que visa descrever esse comportamento (e suas possíveis exceções) de uma maneira mais formal, através de um texto, e que permite ampliar um pouco o conhecimento do mecanismo de operação do software.

A seguir, na Figura 9.6, é exibido o diagrama de casos de uso referente ao sistema CAD que foi desenvolvido.

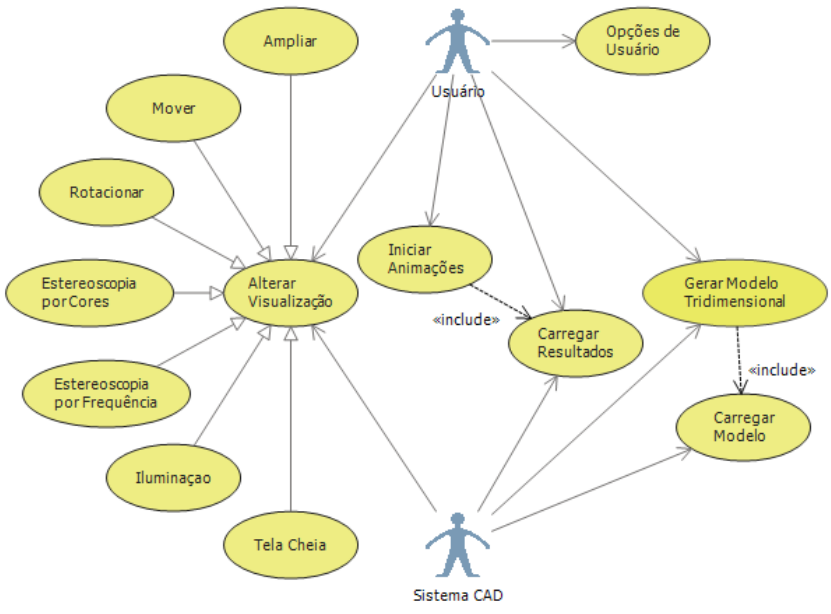

Figura 9.6 - Diagrama de Casos de Uso

9.2.1. Descrição dos Casos de Uso

Os casos de uso são descritos com o seguinte formato:

NOME DO CASO DE USO.

Ator principal:	Quem inicia a ação diretamente.
Atores secundários:	Quem inicia a ação indiretamente.
Objetivo no contexto:	O que se pretende com a execução correta do caso de uso.
Pré-condições:	Condições que já devem existir para que o caso de uso possa acontecer.
Acionamento:	O que leva ao acontecimento do caso de uso.
Cenário:	Passos do caso de uso.
Exceções:	Possíveis exceções que possam ocorrer durante o caso.

Os casos de uso da Figura 9.6 são descritos a seguir.

OPÇÕES DE USUÁRIO

Ator principal:	Usuário
Atores secundários:	Nenhum
Objetivo no contexto:	Alterar opções do programa para maior conforto do usuário
Pré-condições:	Nenhuma
Acionamento:	Alterar as opções desejadas no menu de ferramentas.
Cenário:	O programa está em execução e o usuário seleciona o menu ferramentas. Faz as alterações desejadas e as salva.
Exceções:	Nenhuma

CARREGAR MODELO

Ator principal:	Sistema CAD
Atores secundários:	Nenhum
Objetivo no contexto:	Carregar o modelo bidimensional para futura geração tridimensional do cabo a ser visualizado no programa.
Pré-condições:	O modelo bidimensional deve estar acessível.
Acionamento:	O carregamento dos dados do modelo se dá automaticamente a cada inicialização do modulo de visualização tridimensional.
Cenário:	O programa é iniciado e faz a requisição do modelo. O modelo é carregado na base de dados do programa.
Exceções:	Nenhuma

GERAR MODELO TRIDIMENSIONAL

Ator principal:	Usuário
Atores secundários:	Sistema CAD
Objetivo no contexto:	Gerar o modelo tridimensional.
Pré-condições:	Os dados do modelo do cabo devem ter sido carregados.
Acionamento:	O usuário seleciona o botão '*Alterar para Visualização Tridimensional*'.
Cenário:	O programa é iniciado e os dados do modelo são carregados (*Carregar Modelo*). O usuário seleciona a opção de visualização tridimensional. O modelo tridimensional fica armazenado na memória do programa.
Exceções:	Impossibilidade de gerar o modelo tridimensional devido a inconsistência de dados. Uma mensagem deve ser exibida e o programa deve ser fechado.

CARREGAR RESULTADOS

Ator principal:	Usuário
Atores secundários:	Sistema CAD
Objetivo no contexto:	Carregar dados de um resultado de análise para visualização no protótipo tridimensional.
Pré-condições:	Ao menos um resultado de análises deve ter sido efetuado previamente.
Acionamento:	O usuário carrega um arquivo de resultados de análise.
Cenário:	Com o programa em execução, o usuário seleciona o menu '*Load Analysis*'. Seleciona-se um arquivo com resultados de análises e o arquivo é carregado na base de dados do programa.
Exceções:	Arquivo de análise não correspondente ao modelo carregado. Uma mensagem de erro deve ser mostrada e o carregamento da análise é abortado.

INICIAR ANIMAÇÃO

Ator principal:	Usuário
Atores secundários:	Nenhum
Objetivo no contexto:	Iniciar um módulo de animação, onde é possível visualizar deslocamentos do cabo quando submetido a determinado esforço.
Pré-condições:	Um arquivo de análise de deslocamentos deve ter sido carregado.
Acionamento:	O usuário seleciona a carga do módulo de animação.
Cenário:	Com o programa em execução e ao menos um resultado de análise de deslocamentos carregado, o usuário seleciona o menu *'Iniciar Animação'*. Um menu com as opções de animações é exibido e o usuário pode fazer o controle da forma desejada.
Exceções:	Nenhuma

ALTERAR VISUALIZAÇÃO

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Mudar a visualização atual do cabo. Esta ação engloba as ações denominadas: *Ampliar*, *Mover*, *Rotacionar*, *Estereoscopia por Cores*, *Estereoscopia por Frequência*, *Iluminação* e *Tela Cheia*.
Pré-condições:	O software deve estar em modo de visualização tridimensional.
Acionamento:	O usuário seleciona o tipo de alteração na visualização que deseja.
Cenário:	O programa está em execução e o novo tipo de visualização é aplicado.
Exceções:	Nenhuma

AMPLIAR

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Ampliar ou reduzir o modelo para visualização de detalhes.
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona o grau de ampliação desejado.
Cenário:	O programa está em execução e o usuário seleciona o menu *Zoom*. Seleciona-se o fator pelo qual deseja se aproximar ou afastar da imagem e aplica-se as alterações ao modelo.
Exceções:	Nenhuma

MOVER

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Mudar a posição atual do cabo na janela de visualização.
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona certo deslocamento relativo para a posição do cabo.
Cenário:	O programa está em execução e o usuário seleciona o menu *Pan*. Seleciona-se o deslocamento desejado e aplica-se as alterações ao modelos.
Exceções:	Nenhuma

ROTACIONAR

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Rotacionar o cabo para sua visualização a partir de diversos ângulos diferentes.
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona uma rotação para o cabo.
Cenário:	O programa está em execução e o usuário seleciona o menu para rotacionar o cabo. Seleciona-se o eixo que será rotacionado e o ângulo de rotação. As alterações são aplicadas ao modelo.
Exceções:	Nenhuma

ILUMINAÇÃO

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Mudar o tipo de iluminação para uma melhor visualização do cabo.
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona uma nova iluminação para o cabo.
Cenário:	O programa está em execução e o usuário seleciona uma nova forma de iluminação. A nova iluminação é definida por tipo de fonte de iluminação e cor da fonte luminosa. As alterações são aplicadas ao modelo tridimensional.
Exceções:	Nenhuma

ESTEREOSCOPIA POR CORES

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Iniciar o modo de realidade virtual (estereoscopia por cores).
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona o menu para iniciar estereoscopia por cores.
Cenário:	O programa está em execução e o usuário seleciona o menu de estereoscopia por cores. Um novo modo de visualização é gerado e as alterações são aplicadas ao modelo tridimensional.
Exceções:	Nenhuma

ESTEREOSCOPIA POR FREQUÊNCIA

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Iniciar o modo de realidade virtual (estereoscopia por frequência).
Pré-condições:	A visualização deve estar em tela cheia.
Acionamento:	O usuário seleciona a tecla de atalho para iniciar a estereoscopia por frequência.
Cenário:	O programa está em execução em tela cheia e o usuário seleciona a tecla de atalho para iniciar a estereoscopia por frequência. Um novo modo de visualização é gerado e as alterações são aplicadas ao modelo tridimensional.
Exceções:	Nenhuma

TELA CHEIA

Ator principal:	Usuário
Atores secundários:	CAD System
Objetivo no contexto:	Alternar entre o modo de visualização de janela para tela cheia, e vice-versa.
Pré-condições:	Nenhuma
Acionamento:	O usuário seleciona o menu ou a tecla de atalho para alternar entre o modo tela cheia e janela.
Cenário:	O programa está em execução (em janela ou tela cheia) e o usuário seleciona a opção para alternar o modo de visualização. Um novo modo de visualização é gerado e apresentado ao usuário.
Exceções:	Nenhuma

Capítulo 10. Implementação do Módulo Tridimensional

10.1. Estrutura do Software

O software desenvolvido foi estruturado em diversas bibliotecas (*DLLs*), de a utilizar um modelo mais próximo possível do MVP (*Model – View – Presenter*), onde a base de dados (*model*) não é acessada diretamente pela interface (*view*) e esse relacionamento é gerenciado por uma camada intermediária (*presenter*) (18). A Figura 10.1 mostra o relacionamento entre as diversas bibliotecas criadas.

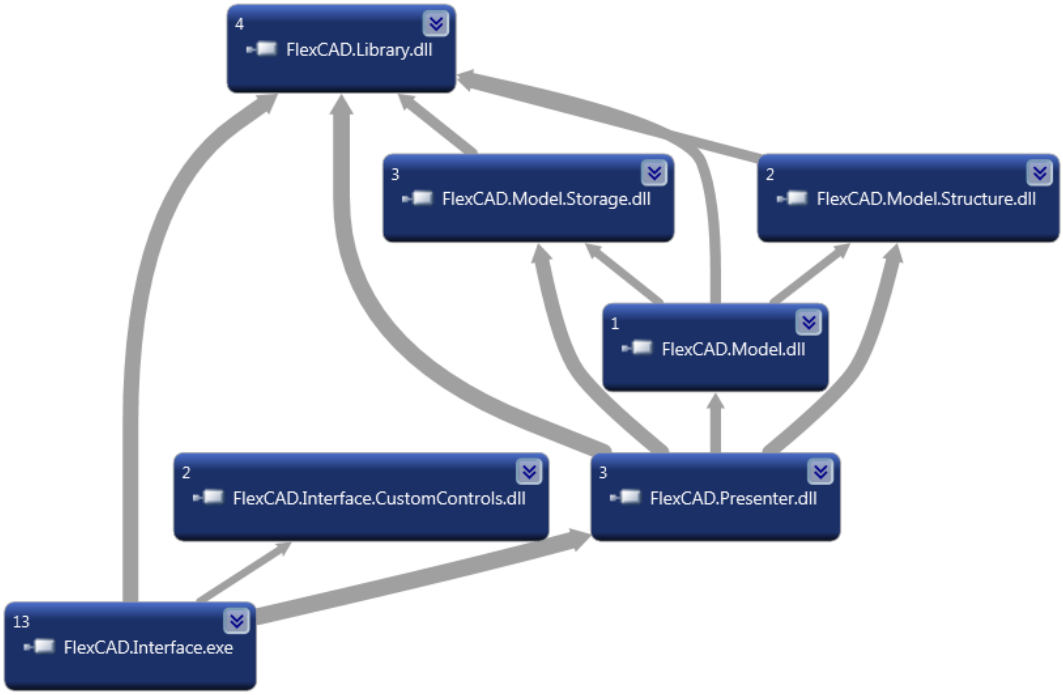

Figura 10.1 - Diagrama de Dependências de Bibliotecas

Primeiramente têm-se a biblioteca *FlexCAD.Library*, que possui itens diversos e comuns a todas as outras bibliotecas, não possui nenhuma lógica interna de processamento, e é referenciada pela demais bibliotecas.

A biblioteca *FlexCAD.Model.Storage* implementa a base de dados responsável pelo repositório do programa, com componentes e materiais, como explicado no Capítulo 2. A

biblioteca *FlexCAD.Model.Structure* é responsável pela implementação da estrutura do cabo, com as camadas e as instâncias. Ambas as bibliotecas, juntamente com a *FlexCAD.Model*, implementam a base de dados do programa.

Esta base de dados é por sua vez utilizada pela biblioteca *FlexCAD.Presenter*, responsável por gerenciar e executar todas as ações do usuário, além de gerar os modelos tridimensionais.

Por último, tem-se a biblioteca *FlexCAD.Interface*, que implementa unicamente uma interface para o usuário, não possuindo nenhuma lógica de processamento, sendo que apenas envia ações para a camada de apresentação (*presenter*) processá-la. Adicionalmente, foi criada a biblioteca *FlexCAD.Interface.CustomControls*, que apenas implementa controles customizados necessários para a interface.

10.2. Geração tridimensional dos componentes

Para a geração tridimensional dos componentes, optou-se pela utilização das técnicas de arrasto e revolução vistas anteriormente. Esta opção foi tomada devido à natureza dos componentes presentes em cabos umbilicais, como as mangueiras por exemplo, que são fabricadas por extrusão e dispostas em helicóides, permitindo a utilização da técnica de arrasto de forma a gerar a sólido de forma simplificada.

Para a geração tridimensional das capas plásticas, foi utilizada a técnica de revolução, e o resultado obtido pode ser verificado na Figura 10.2. Este componente também poderia ter sido gerado a partir de um arrasto, porém, devido ao modelo utilizado para a simulação dos componentes, optou-se por utilizar a revolução, pois esta permite uma maior fidelidade quando o software é utilizado em modo de visualização de resultados de simulação.

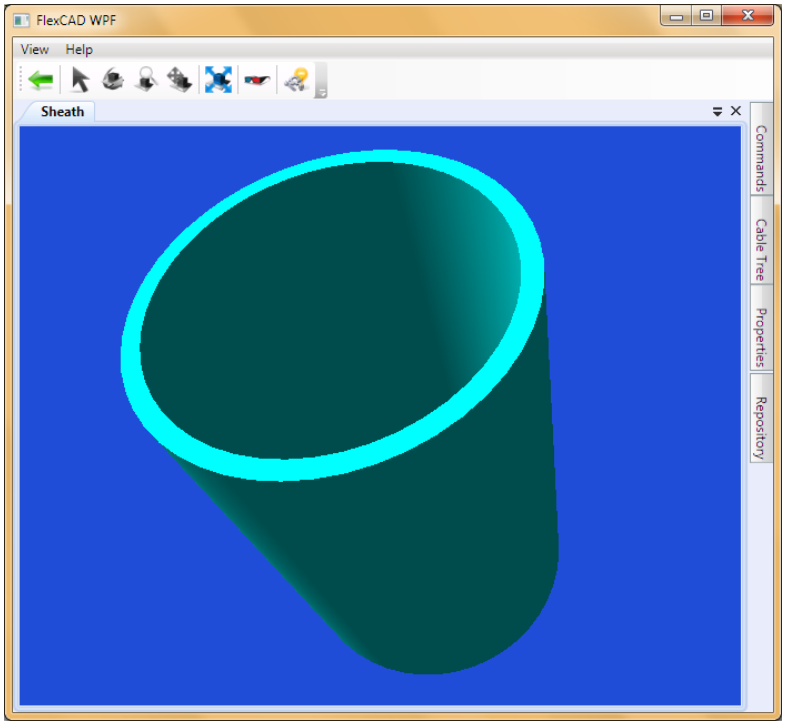

Figura 10.2 - Exemplo de capa plástica gerada por revolução

Para os demais componentes, optou-se pela utilização da técnica de arrasto. O exemplo de uma armadura helicoidal gerada pelo software pode ser visualizada na Figura 10.3.

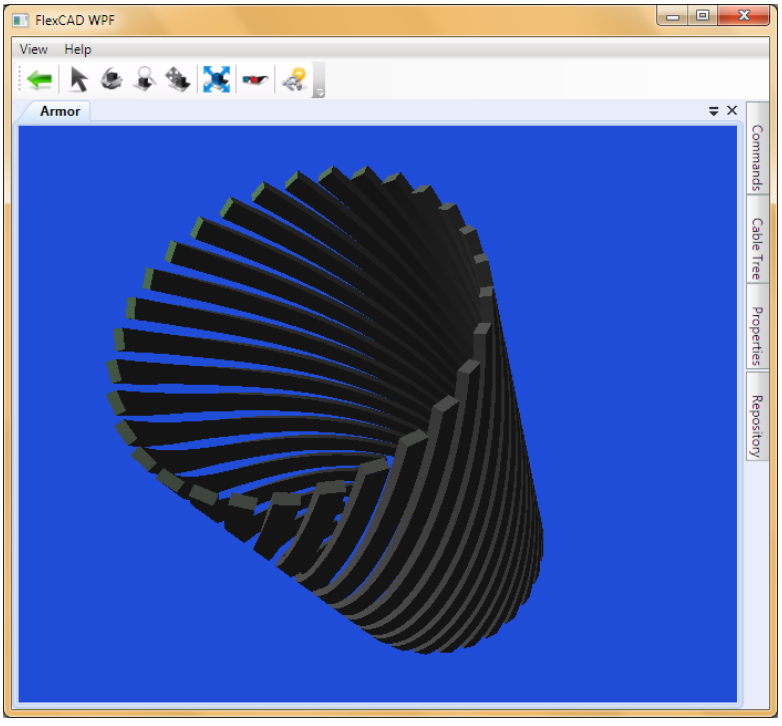

Figura 10.3 - Exemplo de armadura helicoidal gerada por arrasto

No caso das armaduras helicoidais, o perfil utilizado pode ser um retângulo ou um círculo, dependendo do tipo de armadura (tipo este que pode ser customizado pelo módulo bidimensional do software), e deve ser arrastado por um caminho helicoidal. Semelhantemente, a implementação das mangueiras é dada pelo arrasto de um perfil em forma de anel. A Figura 10.4 mostra o resultado gerado por esta operação.

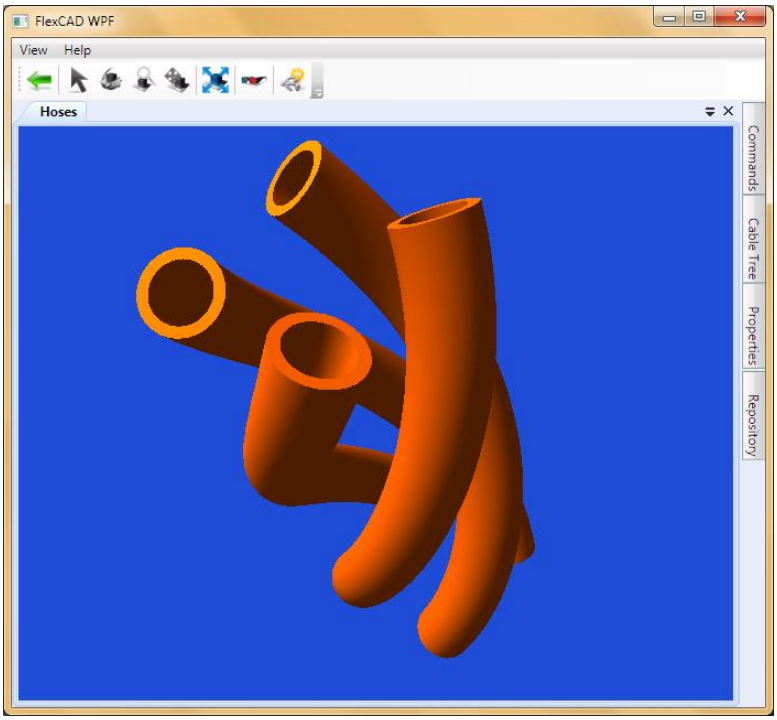

Figura 10.4 - Exemplo de mangueiras geradas por arrasto

Por último, temos os sólidos complexos que englobam carcaças intertravadas e armaduras de pressão. Estes componentes possuem um perfil diferenciado que não é composto simplesmente por primitivas geométricas. A Figura 10.5 mostra um exemplo de perfil de carcaça intertravada.

Figura 10.5 - Exemplo de perfil da carcaça inter-travada

Tanto a carcaça intertravada quanto a armadura de pressão são geradas pelo arrasto de seus respectivos perfis por uma helicoide, da mesma maneira que foi apresentada anteriormente para as mangueiras e armaduras de tração. No entanto, estes sólidos podem ser considerados complexos pois envolvem um grande número de curvas em seu perfil, sendo necessário o cálculo de cada ponto de acordo com o tipo de segmento em que se encontra (reta ou curva), e ainda, com a mesma diferenciação, é necessário o cálculo da normal em cada ponto para que a iluminação se dê de forma realista. A Figura 10.6 mostra um exemplo de carcaça intertravada.

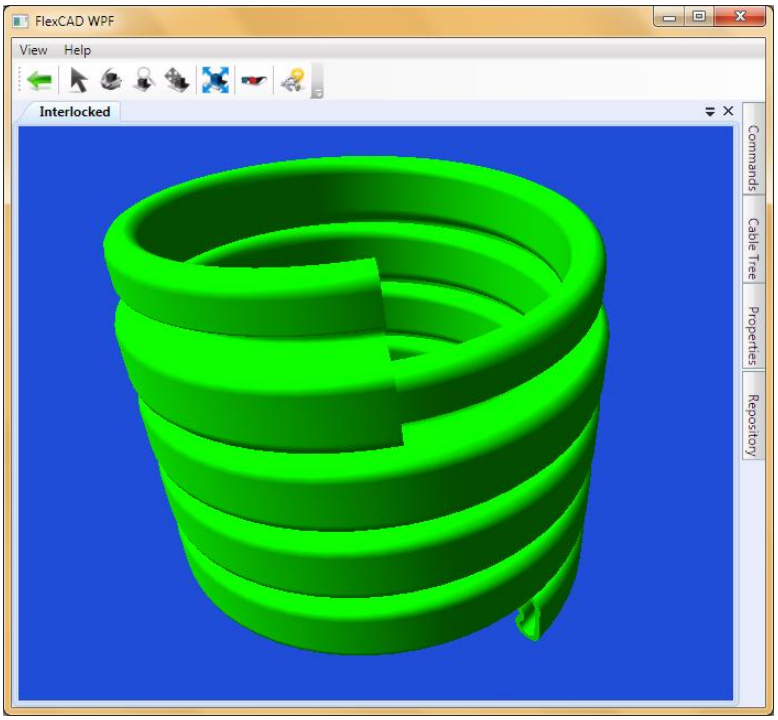

Figura 10.6 - Exemplo de carcaça inter-travada

A Figura 10.7 mostra o exemplo de um cabo eletro-hidráulico gerado pelo software. Neste cabo, estão presentes componentes como mangueiras, armaduras de tração, capas plásticas, preenchimentos e cabos elétricos.

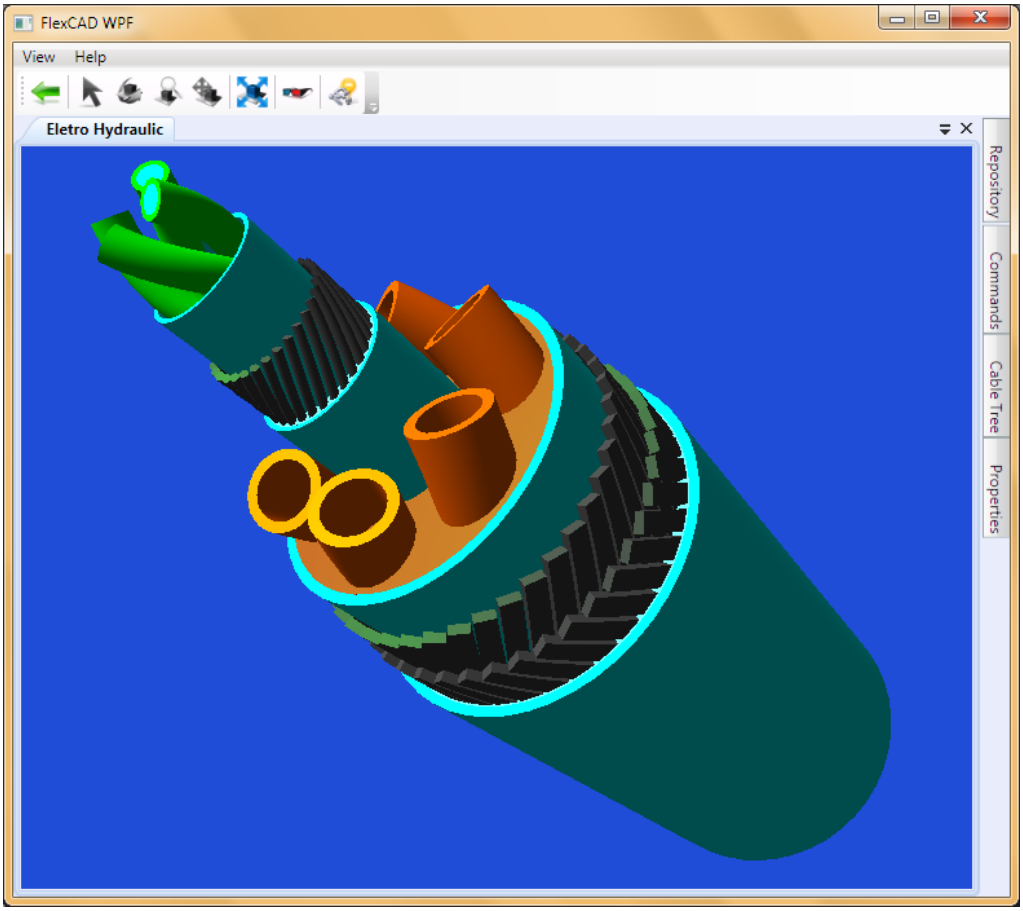

Figura 10.7 - Exemplo de cabo eletro-hidráulico com elementos básicos

10.3. Ferramentas de Visualização

As ferramentas de visualização são utilizadas para permitir ao usuário modificar a visualização da maneira que desejar. Foram implementadas as seguintes ferramentas:

- Rotação
- Translação
- Escala

Outras ferramentas de visualização estão previstas para serem implementadas em fases posteriores de desenvolvimento, como a visualização em estrutura de arames (wireframe), a seleção de componentes, a visualização de vistas pré definidas e a visualização de resultados de simulações.

10.3.1. Rotação

Para a criação de uma ferramenta de rotação que fosse prática e intuitiva para o usuário, decidiu-se pelo emprego da técnica conhecida por '*Virtual Trackball*'.

Com esta técnica, é possível que o usuário rotacione o modelo tridimensional em qualquer direção apenas com o movimento do mouse, ou seja, um movimento bidimensional do mouse é convertido em uma rotação tridimensional.

Isto é feito projetando-se a posição do mouse em uma esfera imaginária que engloba a janela de visualização, como mostrado na Figura 10.8.

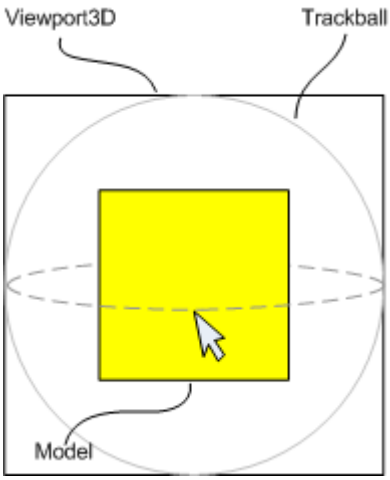

Figura 10.8 - Modelo básico de funcionamento da 'Trackball' – Extraído de (19)

A rotação é calculada impondo-se que a posição do mouse se mantenha constante sobre a esfera, portanto, ao movimentar o mouse, o modelo (ou a câmera) é rotacionado. Por exemplo, ao se movimentar o mouse horizontalmente, uma rotação no eixo Y é necessária para que o mouse se mantenha sobre o mesmo ponto quando projetado na esfera (Figura 10.9).

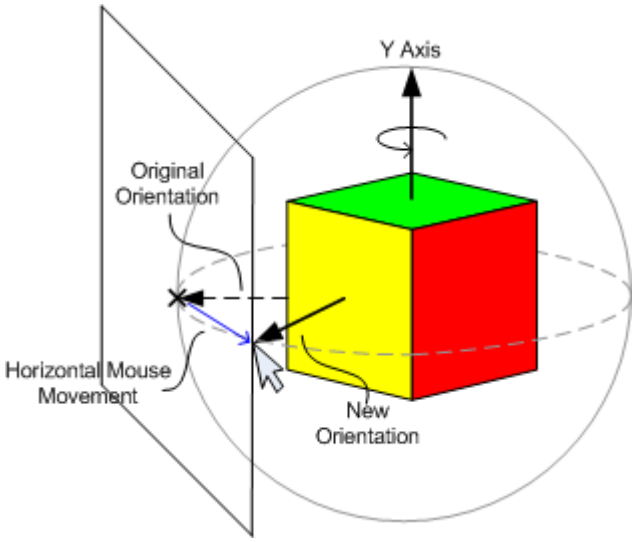

Figura 10.9 - Exemplo de funcionamento da 'Trackball' – Extraído de (19)

São necessários dois passos básicos para que a rotação seja efetuada. Primeiramente é calculado o ponto da esfera que corresponde à posição atual do mouse. Para isto, devemos fazer um ajuste no sistema de coordenadas, uma vez que o elemento gráfico possui um sistema diferente do necessário para a esfera.

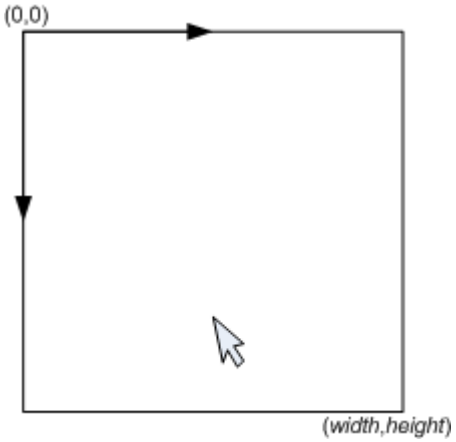

Figura 10.10 - Sistema de coordenadas do elemento gráfico – Extraído de (19)

Na Figura 10.10, podemos ver que o elemento gráfico possui a origem no canto superior esquerdo, sendo seu extremo oposto (canto inferior direito) com coordenadas equivalentes ao seu tamanho (largura, altura).

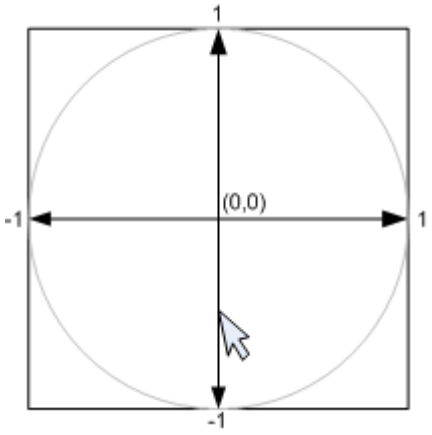

Figura 10.11 - Sistema de coordenadas da esfera – Extraído de (19)

Na Figura 10.11, é representado o sistema de coordenadas da esfera, onde pode ser observado que o sistema tem origem no centro, com coordenadas máximas iguais a um.

Desta forma, a seguinte transformação é necessária:

$$x_{esfera} = \frac{2 \cdot x_{mouse}}{largura} - 1 \qquad 10.1$$

$$y_{esfera} = 1 - \frac{2 \cdot y_{mouse}}{altura} \qquad 10.2$$

Tendo-se as coordenadas x e y do ponto correspondente na esfera, sendo uma esfera de raio igual a um, a coordenada z é dada por:

$$z_{esfera} = \sqrt{1 - {x_{esfera}}^2 - {y_{esfera}}^2} \qquad 10.3$$

Para compor a rotação, é necessário um eixo de rotação e um ângulo. Desta forma, é preciso descobrir qual eixo de rotação e qual ângulo faz com que o mouse permaneça sobre a mesma posição da esfera (Figura 10.12).

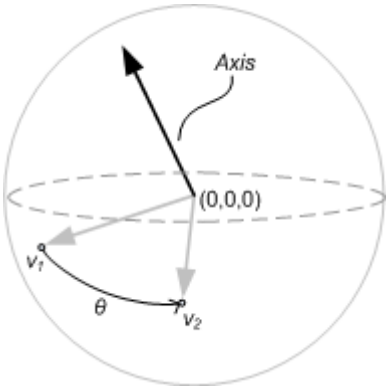

Figura 10.12 - Exemplo para cálculo do eixo e ângulo de rotação

Como a origem da esfera está no centro e o raio é unitário, as posições calculadas anteriormente nos fornecem direções relativas a posição do mouse. Com isso, tendo-se as direções inicias e finais, é possível calcular o eixo de rotação e o ângulo pelas seguintes fórmulas:

$$Eixo = v_1 \times v_2 \qquad 10.4$$

$$\theta = \cos^{-1}\left(\frac{v_1}{|v_1|} \cdot \frac{v_2}{|v_2|}\right) \qquad 10.5$$

Desta forma, basta aplicar esta rotação a transformação já existente do modelo.

10.3.2. Translação

Da mesma maneira que a rotação, decidiu-se pela translação feita a partir dos movimentos do mouse.

Neste caso, um deslocamento do mouse causa igual deslocamento no objeto, ou seja, o ponteiro do mouse permanece sobre a mesma posição do modelo tridimensional. Para isto deve ser associada a coordenada atual do mouse (sistema de coordenadas do elemento gráfico) com a coordenada espacial (sistema de coordenadas do modelo tridimensional).

Primeiramente deve-se transformar a posição do mouse para um ponto correspondente no espaço tridimensional delimitado pela câmera. Desta forma, tem-se casos diferentes para

cada tipo de câmera utilizada. Uma vez que as coordenadas do elemento gráfico são bidimensionais e o espaço visualizado pela câmera é tridimensional, a hipótese de que os pontos estão sempre na coordenada $z_{câmera} = 0$ foi feita.

As coordenadas $x_{câmera}$ e $y_{câmera}$ são calculadas da seguinte maneira:

$$x_{câmera} = \frac{2 \cdot x_{mouse} \cdot xMax}{largura} \quad 10.6$$

$$y_{câmera} = \frac{2 \cdot y_{mouse} \cdot xMax}{altura} \quad 10.7$$

Onde $xMax$ é dependente da câmera utilizada.

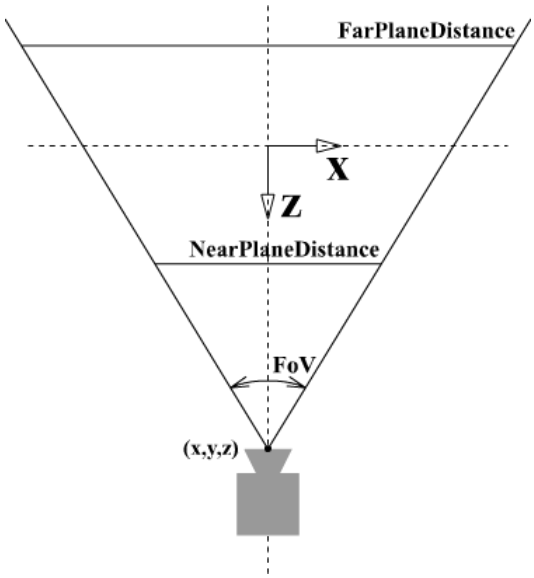

Figura 10.13 - Parâmetros da câmera perspectiva para cálculo de xMax

Para a câmera perspectiva (Figura 10.13), $xMax$ é dado por:

$$xMax = z \cdot \mathrm{tg}\left(\frac{FoV}{2}\right) \quad 10.8$$

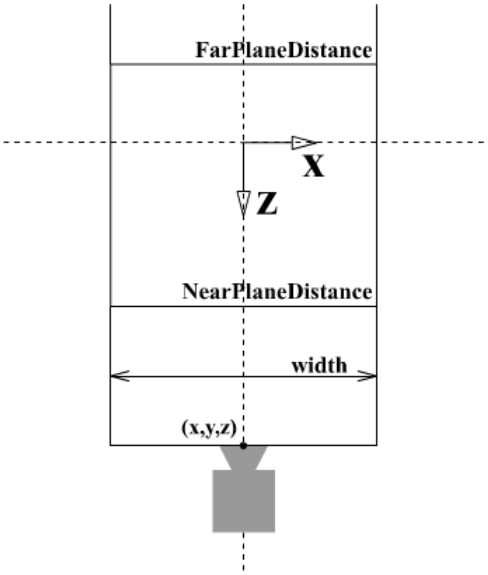

Figura 10.14 - Parâmetros da câmera ortográfica para cálculo de xMax

Para a câmera ortográfica (Figura 10.14), $xMax$ é dado por:

$$xMax = \frac{largura}{2} \qquad 10.9$$

Tendo-se a posição do mouse na coordenada da câmera, basta aplicar a transformação do modelo. Com uma posição inicial e uma posição final, é finalmente calculado o vetor deslocamento a ser aplicado ao modelo para representar o movimento do mouse.

10.3.3. Escala

Para a função de zoom optou-se pela utilização do *scroll* do mouse por sua praticidade, tanto na utilização quanto na implementação.

Existem basicamente dois métodos para esta implementação, sendo o primeiro a movimentação da câmera, aproximando-a do modelo para ampliar e afastando-a para diminuir, e o segundo método é a ampliação do modelo utilizando-se de um fator de escala.

Neste programa, decidiu-se pela utilização de fatores de escala, sendo que cada movimentação do *scroll* do mouse corresponde a um incremento (ou decremento) pré determinado deste fator.

10.4. Iluminação

Como customização de iluminação, forem implementadas as opções de utilização de uma fonte de luz ambiente e de até três fontes de luz direcional. A Figura 10.15 mostra a caixa de diálogo de configuração de iluminação.

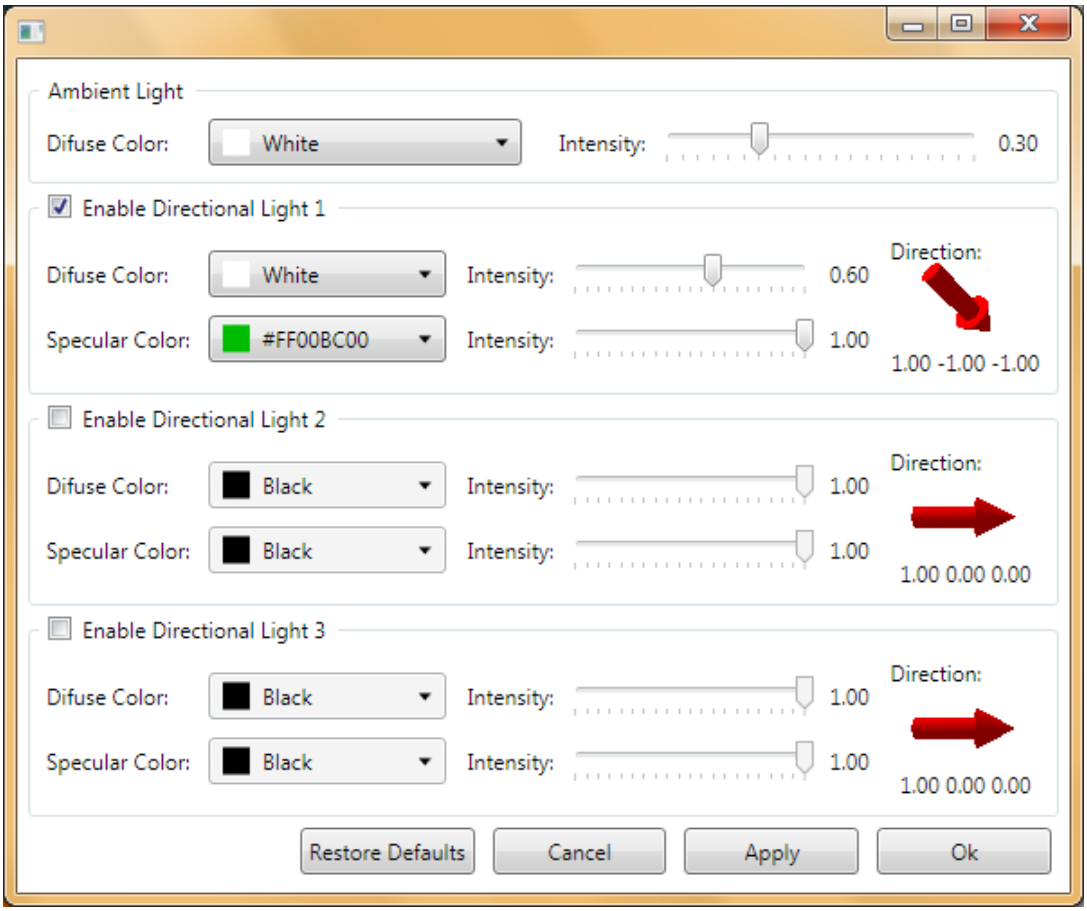

Figura 10.15 - Diálogo de Configuração de Iluminação

A luz ambiente possui a configuração para alteração de sua cor e intensidade. As fontes de luz direcionais, além de poderem ser ativadas ou desativadas, possuem a possibilidade de alterar a cor e a intensidade das componentes difusa e especular separadamente, enquanto que a direção dos raios luminosos pode ser configurada facilmente pela seta no lado direito da caixa de diálogo.

10.5. Texturas

Para a implementação de texturas, foi necessária uma atualização no módulo bidimensional, de forma a permitir que fosse associada uma textura a cada material. A Figura 10.16 mostra como é feita a escolha da textura para aplicação no modelo tridimensional. É importante ressaltar que a seleção de uma textura não causa alterações no desenho bidimensional, que continua sendo desenhado utilizando a cor definida.

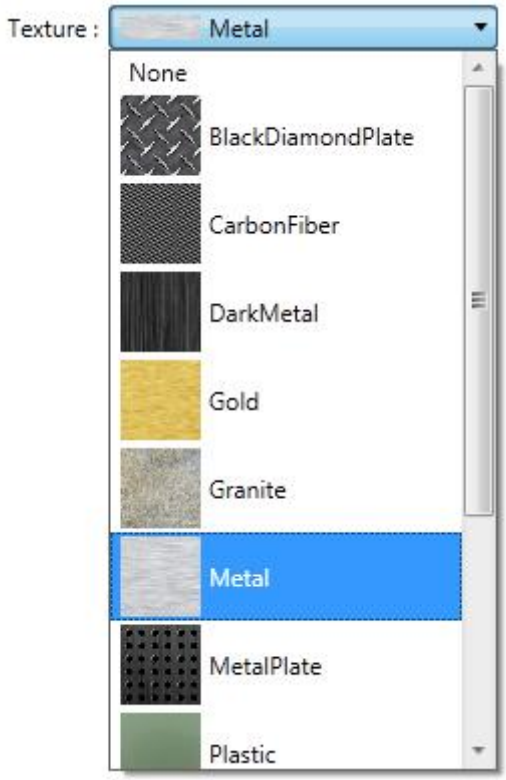

Figura 10.16 - ComboBox de seleção de texturas

A seleção de uma textura não é obrigatória, sendo que nesse caso é utilizada a opção '*None*', que também é a seleção padrão. O software possui texturas que podem ser úteis no projeto de um cabo umbilical ou tubo flexível, como diversos tipos de metais e plásticos.

Ao ser gerado um elemento com uma textura definida, a mesma é mesclada com a cor do material, o que permite ainda uma maior flexibilidade, pois torna-se possível utilizar a

textura com cores diferentes dependendo do desejado, além de uma maior fidelidade à realidade.

A Figura 10.17 mostra um exemplo da geração tridimensional de um tubo flexível projetado pelo módulo bidimensional onde foram selecionadas texturas diversas em seus componentes.

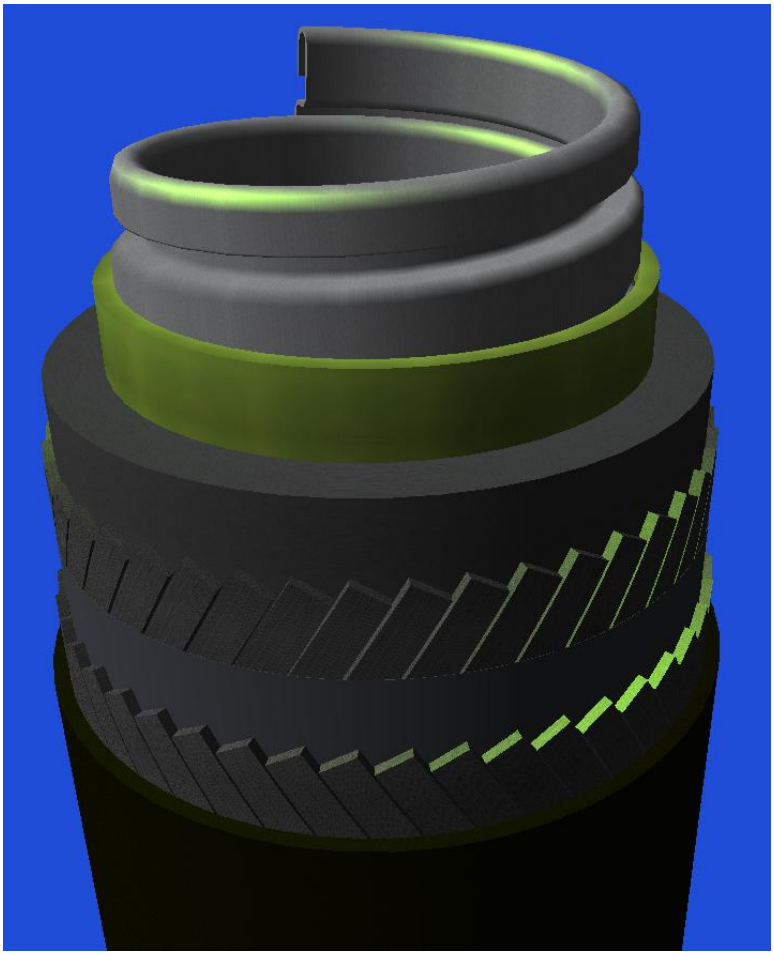

Figura 10.17 - Exemplo de tubo flexível com utilização de texturas

10.6. Realidade Virtual

Por fim, foi implementada a possibilidade de visualização do cabo projetado em modo de realidade virtual, o que, com a utilização de óculos específicos, permite ao usuário uma experiência tridimensional diferenciada, possibilitando a imersão tridimensional e a percepção mais realista de profundidade.

Para esta implementação, foi utilizada a polarização por cores com a combinação Vermelho - Ciano. Desta forma, basta ao usuário utilizar óculos com filtros nestas cores para que possa visualizar corretamente o cabo. A Figura 10.18 mostra um exemplo desta visualização.

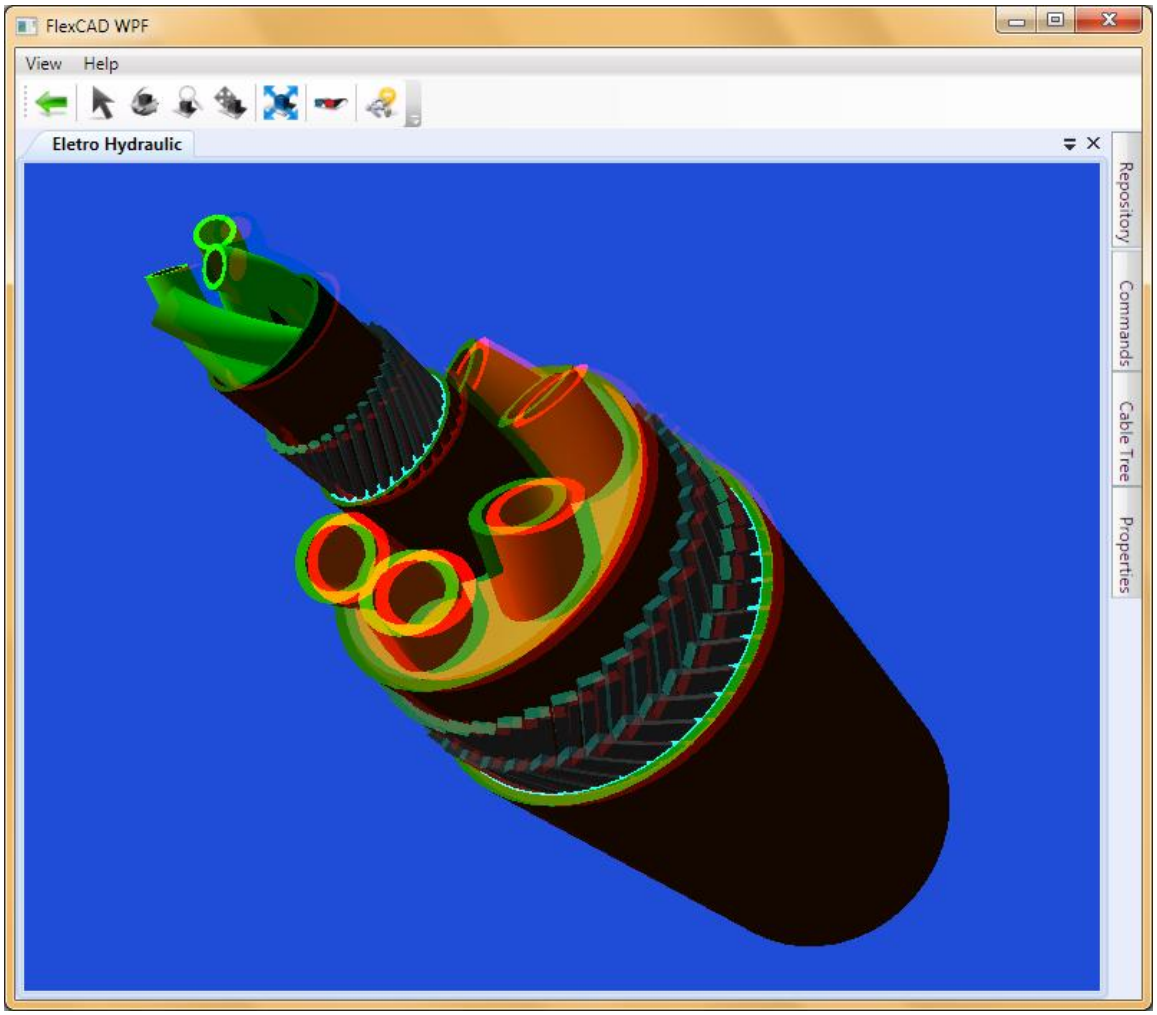

Figura 10.18 - Cabo eletro hidráulico visto em modo de realidade virtual

Capítulo 11. Conclusão

Foi desenvolvida a ferramenta prevista de forma a proporcionar a visualização tridimensional de um cabo umbilical ou tubo flexível a partir do projeto bidimensional do mesmo.

Para tal desenvolvimento, foram estudados diversos conceitos de computação gráfica, iniciando-se com a geração de sólidos tridimensionais, o que resultou na escolha dos métodos de arrasto e revolução para utilização no *software*, seguindo com o estudo de projeções, transformações, iluminação e texturas, concluindo-se com o tópico de realidade virtual para que fosse apresentado ao usuário a maior fidelidade à realidade possível.

Durante o estudo teórico, buscou-se também conhecer como é a implementação de tais tópicos em algumas bibliotecas gráficas disponíveis no mercado, dentre elas, o *OpenGL*, o *WPF* e o *XNA*. Levantando-se os requisitos gráficos do software pretendido e as habilidades de cada biblioteca, concluiu-se que a melhor opção é o *XNA*.

Feito tais estudos e definições, foi dado início o projeto do software propriamente dito, fazendo uso de metodologias de desenvolvimento para a criação de um software de qualidade, o que permite reduzir custos e aumentar a produtividade, uma vez que minimizam o retrabalho, pois proporcionam uma visão global do software e permitem fazer ajustes antes de iniciar o desenvolvimento.

Depois de estudadas algumas metodologias de desenvolvimento, frente às necessidades do projeto e o tempo disponível, concluiu-se que a metodologia de desenvolvimento em espiral seria a melhor opção. Utilizaram-se ainda elementos da modelagem baseada em cenários, como os Casos de Uso, o que proporciona um maior conhecimento da dinâmica e resposta do sistema dado as entradas do usuário, e os Diagramas de Fluxo de Dados, que fornecem uma visão ampla de como a informação flui no sistema,

completando a visão da interação com o usuário. Definido o projeto do software, o mesmo foi implementado utilizando os elementos estudados, com a linguagem de programação C#, integrante do *Microsoft .NET Framework 4.0*.

O *software* desenvolvido permite que o usuário tenha uma visualização mais realista do cabo projetado e permite encontrar erros que não são perceptíveis apenas com a visualização bidimensional. Em um caso prático, foi possível encontrar inconsistências no projeto de um tubo flexível. O ângulo de assentamento especificado fazia com que houvesse sobreposição dos tendões, fato esse que somente pode ser visualizado no caso tridimensional.

Por fim, o software desenvolvido permite outras customizações, como a alterações de fontes luminosas, a utilização de texturas e um modo de realidade virtual implementado por cores, sendo necessária a utilização de óculos especiais (óculos com filtros vermelho e ciano) para utilização de tal funcionalidade.

Capítulo 12. Bibliografia

1. **Prysmian Cables & Systems.** *Umbilical Thermoplastic and Steel Tube.* [Catálogo]. 2007.

2. **Wellstream.** *Manufactures of Spoolable Pipeline Solutions.* [Catálogo]. s.l. : Wellstream International Limited, 2005.

3. **Petzold, Charles.** *3D Programming for Windows®: Three-Dimensional Graphics Programming for the Windows Presentation Foundation.* Washington : Microsoft Press, 2007.

4. **Castanheira, Danilo Balby Silva.** Geometria construtiva de sólidos combinada à representação b-rep. Brasília : Universidade de Brasília, Dezembro de 2003.

5. **Pfreundner, Simona.** Create a 3D Push Pin and a Paper Note in Illustrator Tutorial. *Online Tutorials.* [Online] Julho 31, 2008. http://vector.tutsplus.com/tutorials/illustration/create-a-3d-push-pin-and-a-paper-note-in-illustrator/.

6. **Timóteo, Guilherme T. S. e Monte-Mor, Juliano de A.** Desenho de objetos em OpenGL - Computação Gráfica. [Online] Dezembro de 2010. http://www.dcc.ufla.br/~bruno/aulas/cg/monte-mor/.

7. **Wikipedia.** HSL and HSV - Wikipedia, the free encyclopedia. [Online] 2010. http://en.wikipedia.org/wiki/HSL_and_HSV.

8. **Library, MSDN.** Color Spaces (Windows). [Online] 2010. http://msdn.microsoft.com/en-us/library/dd316799(v=VS.85).aspx.

9. **Wikipedia.** Ray Tracing - Wikipedia, a enciclopédia livre. [Online] 2010. http://pt.wikipedia.org/wiki/Ray_tracing.

10. **NeHe Productions.** Neon Helium Productions. *Nehe GameDev.* [Online] 1997-2010. http://nehe.gamedev.net/.

11. XNA Education Roadmap. *XNA Creators Club.* [Online] http://creators.xna.com/.

12. **Wikipedia.** Bump mapping - Wikipedia, a enciclopédia livre. [Online] 2010. http://pt.wikipedia.org/wiki/Bump_mapping.

13. **Foster, J. Nic.** Normal Mapping Sample. *Nic's GameDev Site.* [Online] Agosto 2007. http://www.nfostergames.com/.

14. *The In and Out: Making Games Play Right with Stereoscopic 3D Technologies.* **Gateau, Samuel.** s.l. : NVidia, 2009. Game Developers Conference - GDC.

15. **Wikipedia.** Stereoscopy - Wikipedia, the free encyclopedia. [Online] 2010. http://en.wikipedia.org/wiki/Stereoscopy.

16. **Silicon Graphics, Inc.** OpenGL Performer™ - Getting Started Guide. 3.2 [Manual]. 2004.

17. **Petzold, Charles.** *Applications = Code + Markup: A Guide to the Microsoft® Windows® Presentation Foundation.* Washington : Microsoft Press, 2006.

18. **Boodhoo, Jean-Paul.** Desing Patterns - Model View Presenter. *MSDN Magazine.* [Online] 2010. http://msdn.microsoft.com/en-us/magazine/cc188690.aspx.

19. **Lehenbauer, Daniel.** Rotating the Camera with the Mouse. *Implementing a Virtual Trackball with the Windows Presentation Foundation (formerly codenamed Avalon).* [Online] http://viewport3d.com/trackball.htm.

20. **Marshall, Donis.** *Programming Microsoft Visual C# 2008: The Language.* s.l. : Microsoft Press, 2008.

21. **Wangenheim, Aldo von.** Iluminação - Visualização Realística em 3D, zbuffering e Raytracing. [Online] http://www.inf.ufsc.br/~awangenh/CG/raytracing/iluminacao.html.

22. **Gattass, Marcelo.** Computação Gráfica Interativa. [Online] 2010. http://www.tecgraf.puc-rio.br/~mgattass/cg/cg.html.

23. **Liberty, Jesse e Xie, Donald.** *Programming C# 3.0.* 5ª Edição. s.l. : O'Reilly, 2008.

24. **Wikipedia.** Anaglyph image - Wikipedia, the free encyclopedia. [Online] 2010. http://en.wikipedia.org/wiki/Anaglyph_image.

25. The CMYK, RGB and HSB Color Models. *Web Builder's Café.* [Online] 2010. http://webbuilderscafe.com/color_models.htm.

26. **Library, MSDN.** Transforms (Direct3D 9) (Windows). [Online] 2010. http://msdn.microsoft.com/en-us/library/bb206269(v=VS.85).aspx.

27. **Teixeira, Fábio Gonçalves.** *Modelamento Paramétrico e Geração de Malha em Superfícies para Aplicação em Engenharia.* Porto Alegre : s.n., 2003.