**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIAS DA COMPUTAÇÃO**

(Bacharelado)

# PROTÓTIPO DE FERRAMENTA/PLUG-IN PARA GERAÇÃO DE IMAGENS 3D A PARTIR DE IMAGENS RASTER 2D EM GRAYSCALE PARA O PHOTOSHOP®

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

**RICARDO FACHINI**

BLUMENAU, JUNHO/2000

2000/1-57

# PROTÓTIPO DE FERRAMENTA/PLUG-IN PARA GERAÇÃO DE IMAGENS 3D A PARTIR DE IMAGENS RASTER 2D EM GRAYSCALE PARA O PHOTOSHOP®

**RICARDO FACHINI**

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

**BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO**

Prof. Antônio Carlos Tavares — Orientador na FURB

Prof. José Roque Voltolini da Silva — Coordenador do TCC

**BANCA EXAMINADORA**

Prof. Antônio Carlos Tavares

Prof. Dalton Solano dos Reis

Prof. José Roque Voltolini da Silva

# DEDICATÓRIA

Dedico este trabalho a minha namorada Sabrina,

aos meus familiares e amigos

e principalmente a meus pais Mário e Vergínia,

pelo apoio dado durante a elaboração deste trabalho.

# AGRADECIMENTOS

Ao Professor Antônio Carlos Tavares, pela paciência e pelo interesse com o qual orientou este trabalho.

Ao Professor Dalton Solano dos Reis, pelos esforços dispensados no decorrer do semestre, auxiliando assim o desenvolvimento deste trabalho.

Ao Professor José Roque Voltolini da Silva, coordenador do Trabalho de Conclusão de Curso.

A todos os professores e funcionários do Departamento de Sistemas e Computação que auxiliaram para que este trabalho pudesse ser realizado.

Ao colega Francisco Beltrão, pelo apoio recebido e pela contribuição com o ensinamento da linguagem Visual C++.

Aos colegas, tanto aqueles que ficaram no decorrer do curso, como aos que conseguiram junto comigo chegar ao fim de mais uma etapa de nossas vidas.

E a todos que de alguma forma contribuíram para a realização deste trabalho.

# SUMÁRIO

# LISTA DE FIGURAS

# LISTA DE QUADROS

# LISTA DE TABELAS

# LISTA DE ABREVIATURAS

API - *Aplication Interface*

CG - COMPUTAÇÃO GRÁFICA

CMYK - *CYAN-MANGENTA-YELLOW-BLACK*

GDI - *Graphic Display Interface*

IDE - *Integrated Development Environment*

MFC - *Microsoft Foundation Class*

NDCS - *Normalized Device Coordinate System*

OLE - *Object Linking and Embedding*

PDCS - *Physical Device Coordinate System*

PiPL - *Plug-in Property List*

RGB - *RED-GREEN-BLUE*

# RESUMO

Este trabalho visa a realização de um protótipo que apresentará a criação de um *plug-in* para *Photoshop* e um estudo sobre a conversão de imagens *raster* 2D em *grayscale* para imagens em 3D, o qual implementará técnicas de câmera sintética em ambiente 3D utilizando para isso a biblioteca gráfica *OpenGL*.

# ABSTRACT

This work seeks the accomplishment of a prototype that will present the creation of a plug-in for Photoshop (and a study on the conversion of images raster 2D in grayscale for images in 3D, which it will implement techniques of synthetic camera in environment 3D using for that the OpenGL graphic library.

# 1 INTRODUÇÃO

Observando o atual mercado da área gráfica, verifica-se que muitos profissionais que trabalham com CG, assim como profissionais que necessitam de imagens tridimensionais de fácil e rápido desenvolvimento, para serem utilizadas principalmente para trabalhos gráficos para a internet, dificilmente encontram no mercado uma solução barata e simples para a geração das mesmas.

Para auxiliar esses profissionais foi desenvolvido um protótipo de um *plug-in* para o programa gráfico *Photoshop*, por este ser um software amplamente utilizado pelos profissionais já mencionados e por possibilitar a inclusão de *plug-ins* externos ao mesmo.

Designa-se *plug-in* uma ferramenta incorporada ao *Photoshop*, e a outros programas, visando à inclusão de novos métodos e técnicas para aprimorar, facilitar e incluir novos recursos ao respectivo software ([ALS1995]).

O *Photoshop* é um programa gráfico que trabalha essencialmente com imagens 2D, podendo converter estas imagens em diversos formatos: RGB, CMYK, *grayscale* (imagens que podem chegar a ter 256 tons de cinza) etc ([MCC1999]). O protótipo proposto trabalha somente com imagens em *grayscale*.

O *Photoshop*, por se tratar de um aplicativo para necessidades genéricas em editoração gráfica de ambientes bidimensionais, não permite ter-se um ambiente apropriado para a geração e manipulação de imagens tridimensionais ([MCC1999]).

A implementação tem por objetivo proporcionar como resultado a visualização da imagem original, bidimensional *grayscale* transformada em tridimensional, conforme a escala de tons de cinza presentes na mesma; também permite a inspeção por parte do usuário de uma forma dinâmica através de uma interface amigável, que possibilita um caminhamento ao redor do objeto (gráfico) proporcionando vários ângulos de visão. Para o desenvolvimento desta ferramenta utilizou-se a biblioteca gráfica *OpenGL*.

Para a representação da superfície tridimensional foram utilizados vetores perpendiculares a imagem *raster*. Sendo que cada ponto (*pixel*) desta, terá um vetor com a altura igual ao valor deste ponto. Como resultado ter-se-á uma aproximação da superfície tridimensional, o qual dependerá da resolução e do contexto da imagem.

Este trabalho apresenta um estudo sobre a criação de *plug-ins* para o *Photoshop* utilizando o ambiente de programação *Visual C++ 6.0*, bem como a implementação, para demonstração, de um *plug-in* que proporciona a geração de imagens 3D a partir de imagens raster 2D em *grayscale*.

## 1.1 OBJETIVO

O objetivo principal do trabalho proposto é desenvolver um *plug-in*, para o programa gráfico *Photoshop*; e para demonstração deste *plug-in*, será utilizado um algoritmo que permita a geração de imagens 3D a partir de uma imagem 2D *raster* em *grayscale*, num ambiente tridimensional criado na biblioteca gráfica *OpenGL*.

## 1.2 ORGANIZAÇÃO DO TEXTO

O primeiro capítulo fornece uma introdução ao trabalho desenvolvido, demonstrando qual o objetivo do trabalho e apresentando os principais tópicos deste trabalho.

O segundo capítulo demonstra o que é o *Photoshop* e qual a utilidade do mesmo.

O terceiro capítulo traz uma base teórica sobre *plug-ins*, uma breve história, quais os tipos e extensões dos mesmos.

O quarto capítulo fornece uma visão das técnicas de CG e o formato de imagem utilizada no desenvolvimento do protótipo.

No quinto capítulo é apresentada a biblioteca gráfica *OpenGL*, o que é, o que pode fazer. Ainda, são apresentados alguns comandos básicos e como o *OpenGL* trata a visualização e as transformações geométricas.

O sexto capítulo fornece uma introdução ao *Visual C++*, trazendo algumas das características básicas do ambiente de programação.

No sétimo capítulo são apresentadas as especificações do protótipo, englobando o seu funcionamento e aspectos de implementação.

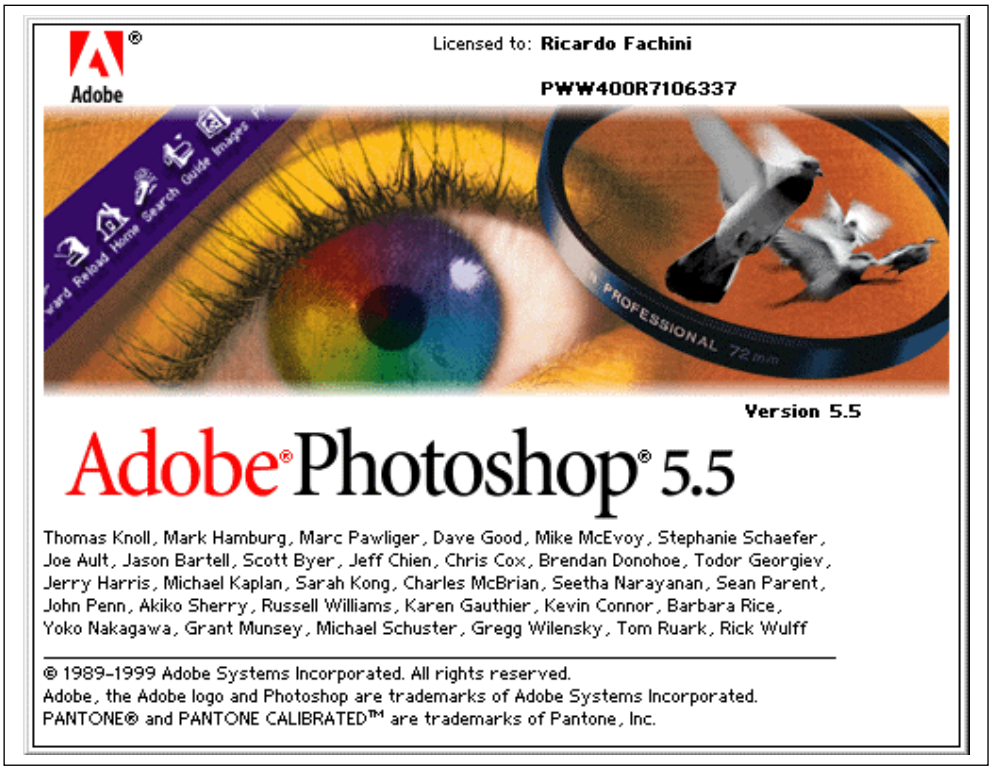
O oitavo capítulo faz uma análise conclusiva sobre o trabalho, dificuldades encontradas, e futuras extensões do trabalho que poderão ser realizadas.

# 2 PHOTOSHOP

## 2.1 O QUE É O PHOTOSHOP

Segundo McClelland ([MCC1999]), o programa gráfico *Adobe Photoshop* é o aplicativo para edição de imagens mais utilizado em computadores *Macintosh* e *Windows*, atualmente. Apesar da concorrência acirrada entre os programas, como *Macromedia xRes*, *Live Picture*, *Writght Design*, entre outros, a *Adobe Systems, Inc*. ([ADO2000]) informa que o *Photoshop* detém mais de 80% das vendas no mercado de edição de imagens (essa estimativa inclui o *Painter* da *MetaCreations*, um programa de criação de imagens que não é concorrente direto do *Photoshop*). Isso torna o *Photoshop* quatro vezes mais usado do que todos os concorrentes juntos. A figura 1 apresenta a tela de abertura do *Photoshop*.

**Figura 1 – Tela de abertura do Photoshop**




A vantagem de vendas sempre superior do *Photoshop* vem sendo para a *Adobe* um incentivo para reinvestir no programa e regularmente aperfeiçoar – e até, em alguns casos, corrigir – seus recursos. Embora os concorrentes possam oferecer recursos

interessantes, o somatório de suas partes ainda permanece inferior ao *Photoshop* ([MCC1999]).

Como resultado, o *Photoshop* mantém o domínio em seu ramo de mercado. Ele nem sempre foi o melhor editor de imagens, nem foi o primeiro deles, mas sua interface objetiva fizera dele o melhor recurso desde a primeira versão. Quase uma década depois – graças a injeção substancial de capital e a uma programação altamente criativa por parte da equipe da *Adobe* e do criador do *Photoshop*, Thomas Knoll, passou a ser o programa mais conhecido de sua categoria ([ADO2000]).

## 2.2 QUANDO UTILIZAR O PHOTOSHOP

Graças a seus métodos especializados, os programas de pintura e de desenho atendem a finalidades distintas e divergentes. O *Photoshop* e outros programas de pintura são mais adequados à criação e edição dos seguintes tipos de arte-final ([MCC1999]):

a) fotos digitalizadas, entre elas colagens fotográficas e ornamentações que se originam nas digitalizações;
b) imagens capturadas com qualquer tipo de câmera digital;
c) artes realísticas compostas por realces naturalistas, meios-tons e sombras;
d) arte em estilo impressionista e outras imagens criadas para fins puramente pessoais e estéticos;
e) logotipos e outros tipos de imagens, utilizando arestas suaves, reflexos e sobras cônicas;
f) efeitos especiais que exigem o uso de filtros e o aperfeiçoamento das cores.

## 2.3 VISÃO GERAL DOS FORMATOS DE ARQUIVO

O *Photoshop* suporta mais de 20 formatos de arquivo nas caixas de diálogo "*Open*" e "*Save*". Ele pode suportar muito mais com a adição de *plug-ins*, que ligam os comandos aos submenus "*Save As*", "*Import*" e "*Export*" (submenu de "*File*").

Os formatos de arquivo representam maneiras diferentes de salvar um arquivo em disco. Alguns formatos proporcionam esquemas únicos de compactação de imagem, que salvam uma imagem de uma maneira que consome menos espaço em disco.

Outros formatos permitem ao *Photoshop* fazer intercâmbio de imagens com diferentes aplicativos executados no Mac, no Windows ou em outras plataformas.

Como a maioria dos programas, o *Photoshop* oferece seu próprio formato nativo, isto é, um formato otimizado para as capacidades e funções particulares do *Photoshop*. Esse formato (.psd) salva cada atributo que você pode aplicar no *Photoshop*, incluindo *layers*, canais adicionais, informações de arquivo e assim por diante.

# 3 MÓDULOS DE PLUG-IN

## 3.1 BREVE HISTÓRIA

*Plug–ins* não são exclusivos do *Adobe Photoshop*. Muitas aplicações para *Macintosh* e *Windows* suportam alguma forma de extensões de *plug–in*.

Uma das primeiras companhias a incorporar módulos de *plug–in* nos seus produtos é a *Silicon Beach*, em seu *Digital Darkroom* e produtos do *SuperPaint* ([ADO2000]).

No início, o *Adobe Photoshop* implementou módulos de *plug–in* semelhante a arquitetura utilizada pela *Silicon Beach*. Porém, a semelhança não durou muito tempo. Com a arquitetura de *plug–ins* evoluída, a interface detalhada para os módulos de *plug–in* do *Photoshop* tornou-se completamente diferente da utilizada pela *Silicon Beach*. As diferenças eram exigidas para suportar imagens coloridas e principalmente o apoio à memória virtual implementado no *Adobe Photoshop*.

## 3.2 MÓDULOS E SERVIDORES DE PLUG-IN

Módulos de *plug–in* são programas desenvolvidos pela *Adobe Systems* e vendedores especializados da *Adobe Systems* para estender uma aplicação. *Plug–ins* podem ser somados ou podem ser atualizados independentemente para personalizar os servidores de *plug-in* para as necessidades particulares.

Um servidor de *plug–in* é responsável por carregar módulos de *plug–in* na memória e os chamar.

Os programas da *Adobe* que funcionam como servidores de *plug-in* são: *Adobe After Effects*, *Adobe Premiere*, *Adobe Illustrator*, *Adobe PageMaker*, *Adobe PhotoDeluxe* e *Adobe Photoshop*. A maioria destes programas suporta alguns, mas não todos os *plug–in* do *Photoshop* ([ADO2000]).

A maioria dos servidores de *plug–in* é um programa, mas isto não é uma exigência. Um servidor de *plug–in* pode ser também um módulo de *plug–in*. Um bom exemplo disto é o "*Photoshop Adapter*", um *plug-in* que permite o *Adobe Ilustrator 6.0* ser servidor de módulos de formato e de filtro do *Photoshop*.

No *Photoshop*, os arquivos dos *plug-ins* estão localizados na pasta "*plug-ins*", dentro da pasta de instalação do *Photoshop*. Para adicionar novos *plug-ins* ao programa, basta adicionar o *plug-in* desejado na respectiva pasta.

No menu de ajuda do *Photoshop* pode-se obter maiores informações sobre os *plug-ins* instalados.

## 3.3 TIPOS DE PLUG-INS

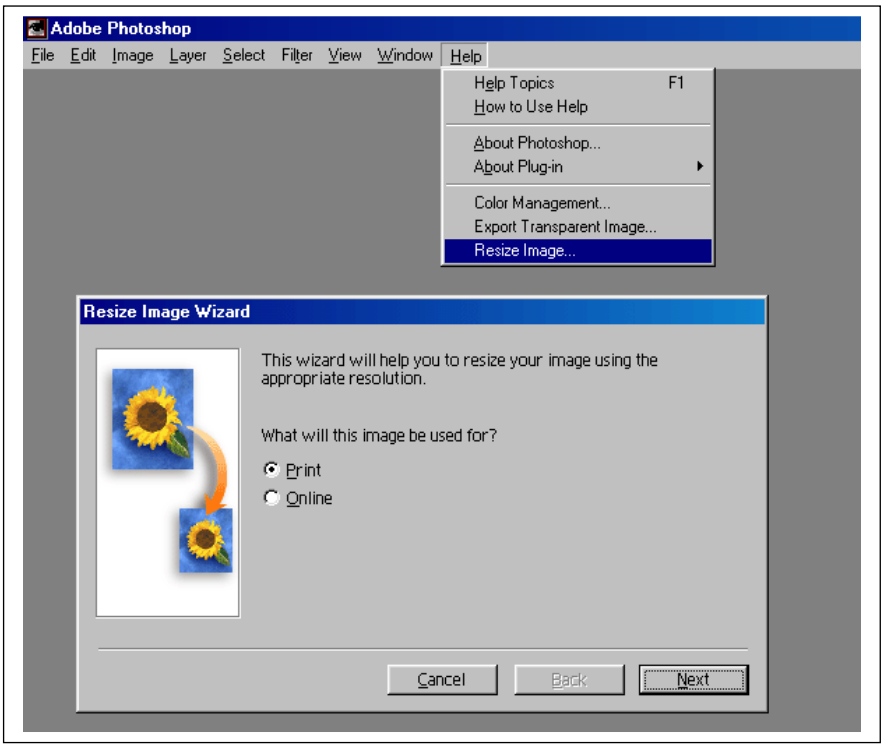
Os módulos de *plug-in* para *Adobe Photoshop* são arquivos externos que contêm código para extender o *Photoshop* sem modificar a aplicação básica.

O *Photoshop* suporta nove tipos de módulos de *plug–ins*, os quais serão vistos adiante.

### 3.3.1 AUTOMATIZAÇÃO (AUTOMATION)

O módulo de automatização tem acesso a todos *scripts* de eventos do *Photoshop*. O *Photoshop*, quando adquirido, geralmente possui os seguintes módulos de automatização: "*Color Management*" (gerenciador de cores para os monitores), "*Export Transparent Image*" (permite exportar imagens com fundo transparente) e "*Resize Image*" (permite aumentar ou diminuir a imagem para a impressão). Este módulo está localizado no menu de ajuda do *Photoshop* (figura 2).

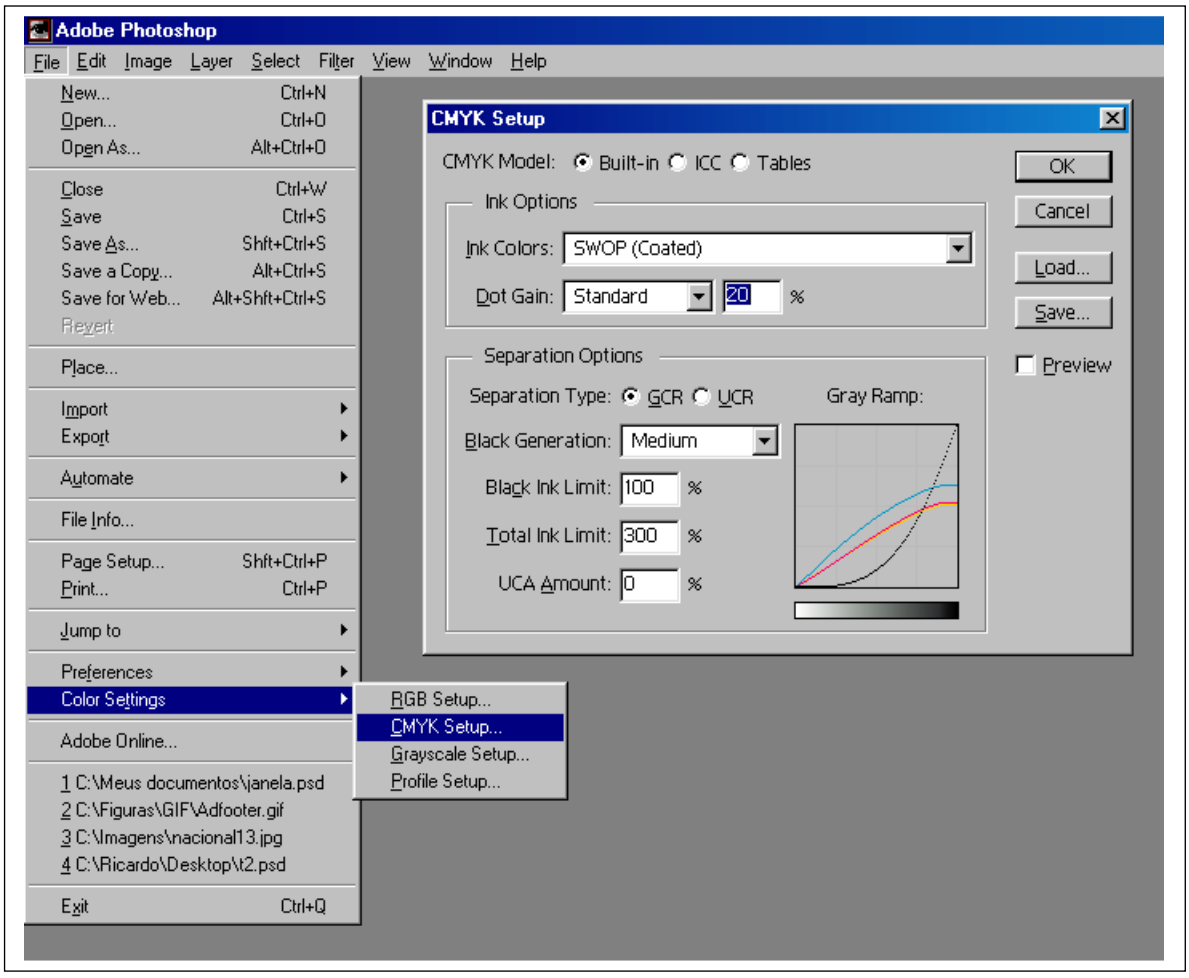
**Figura 2 – Automation Plug-in**



### 3.3.2 PALETA DE CORES (COLOR PICKER)

Este módulo provêm uma interface para seleção de cores diferente da seleção de cores padrão do *Photoshop*. Os módulos de paleta de cores estão localizados no menu "*File*" em "*Color Settings*" (figura 3).

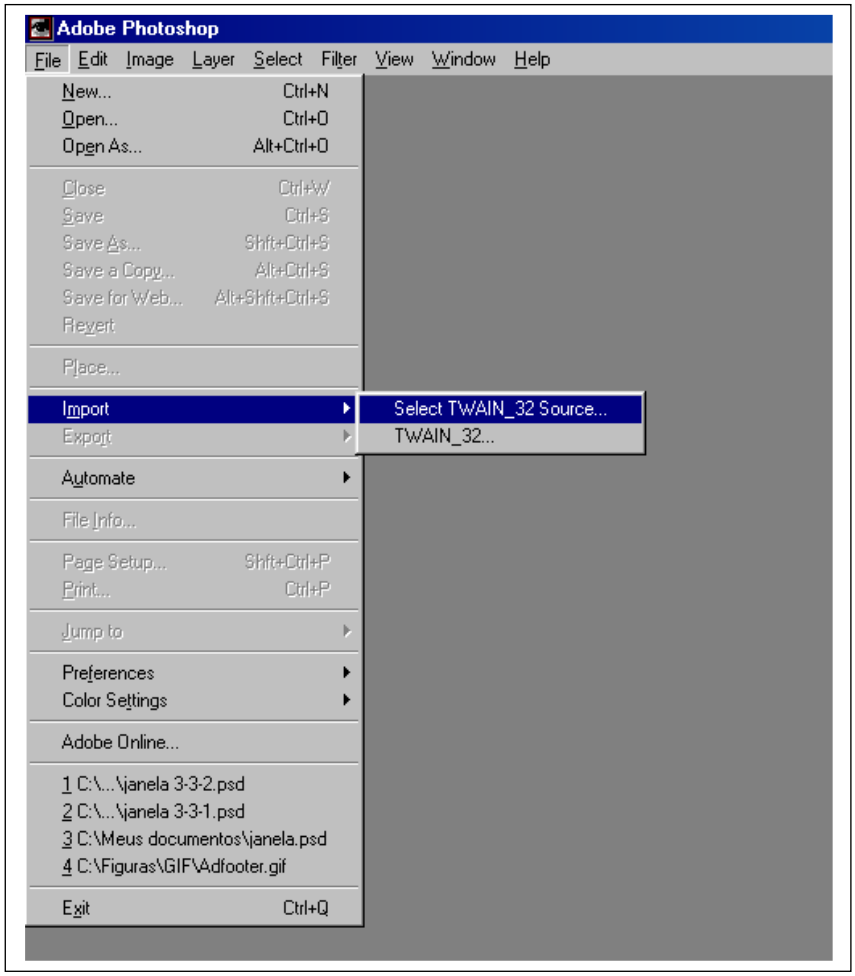
**Figura 3 – Color Picker Plug-in**



### 3.3.3 IMPORTAÇÃO (IMPORT)

Módulos de importação permitem obter uma imagem de um aplicativo externo. Um exemplo de importação é o de obtenção de imagens através de um *scanner*. Este *plug-in* pode ser encontrado no menu "*File*" (figura 4).

**Figura 4 – Import Plug-in**




### 3.3.4 EXPORTAÇÃO (EXPORT)

Módulos de exportação podem ser usados para salvar imagens em formatos de arquivos não suportados pelo *Photoshop* padrão. Este módulo encontra-se no menu "*File*".

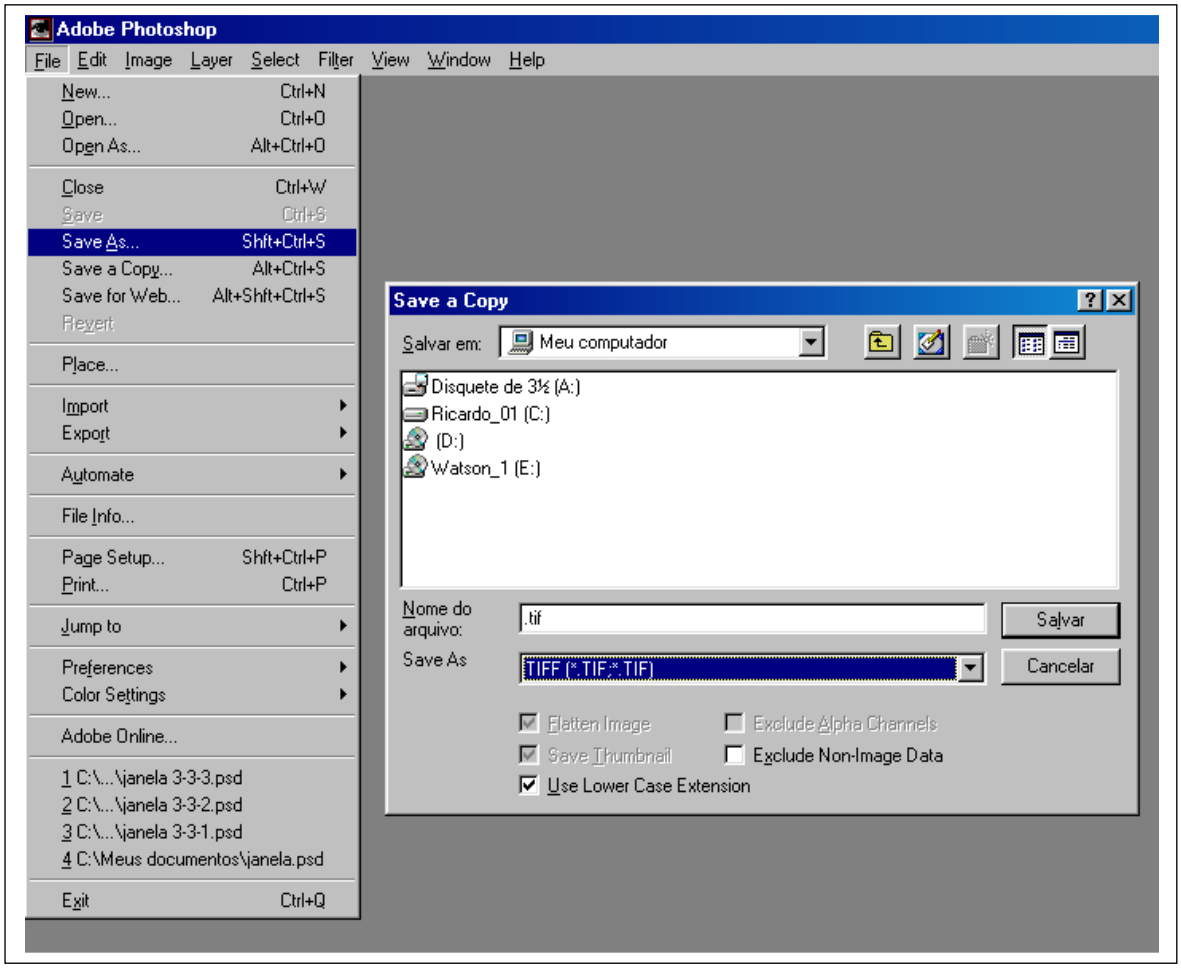
### 3.3.5 EXTENSÃO (EXTENSION)

Módulos de extensão permitem a implementação no início da execução do *Photoshop* e no término da execução do mesmo. Normalmente este módulo não possui nenhuma interface com o usuário.

### 3.3.6 FORMATO (FORMAT)

Formato de arquivo, também chamado de módulo de formato de imagem, provêm apoio para ler e salvar formatos de imagem adicionais. Este módulo encontra-se no menu *"File"* em *"Open"*, *"Save as"* e *"Save as copy"* (figura 5).

**Figura 5 – Format Plug-in**



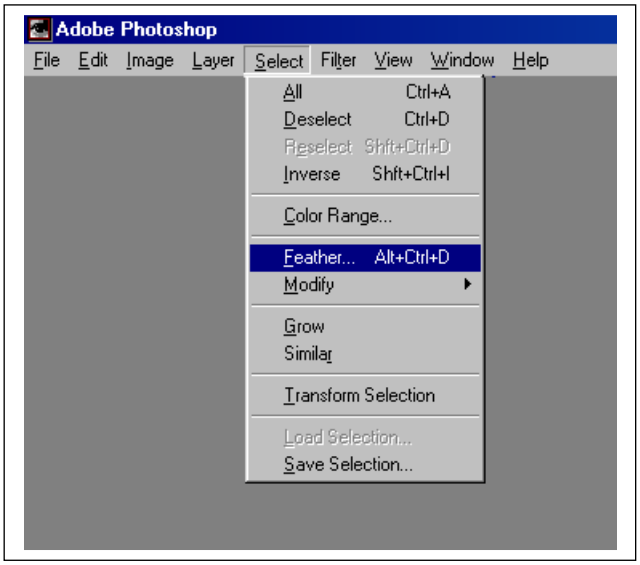
### 3.3.7 PARSER

Módulos de *parser* são semelhantes aos módulos de importação e exportação, e proveêm apoio para dados de manipulação entre o *Photoshop* e outros programas que suportam outros formatos de arquivos (normalmente formato vetorial).

### 3.3.8 SELEÇÃO (SELECTION)

Módulos de seleção modificam o estilo de seleção do *Photoshop*. Estes módulos estão presentes no menu *"Selection"* (figura 6).

**Figura 6 – Selection Plug-in**



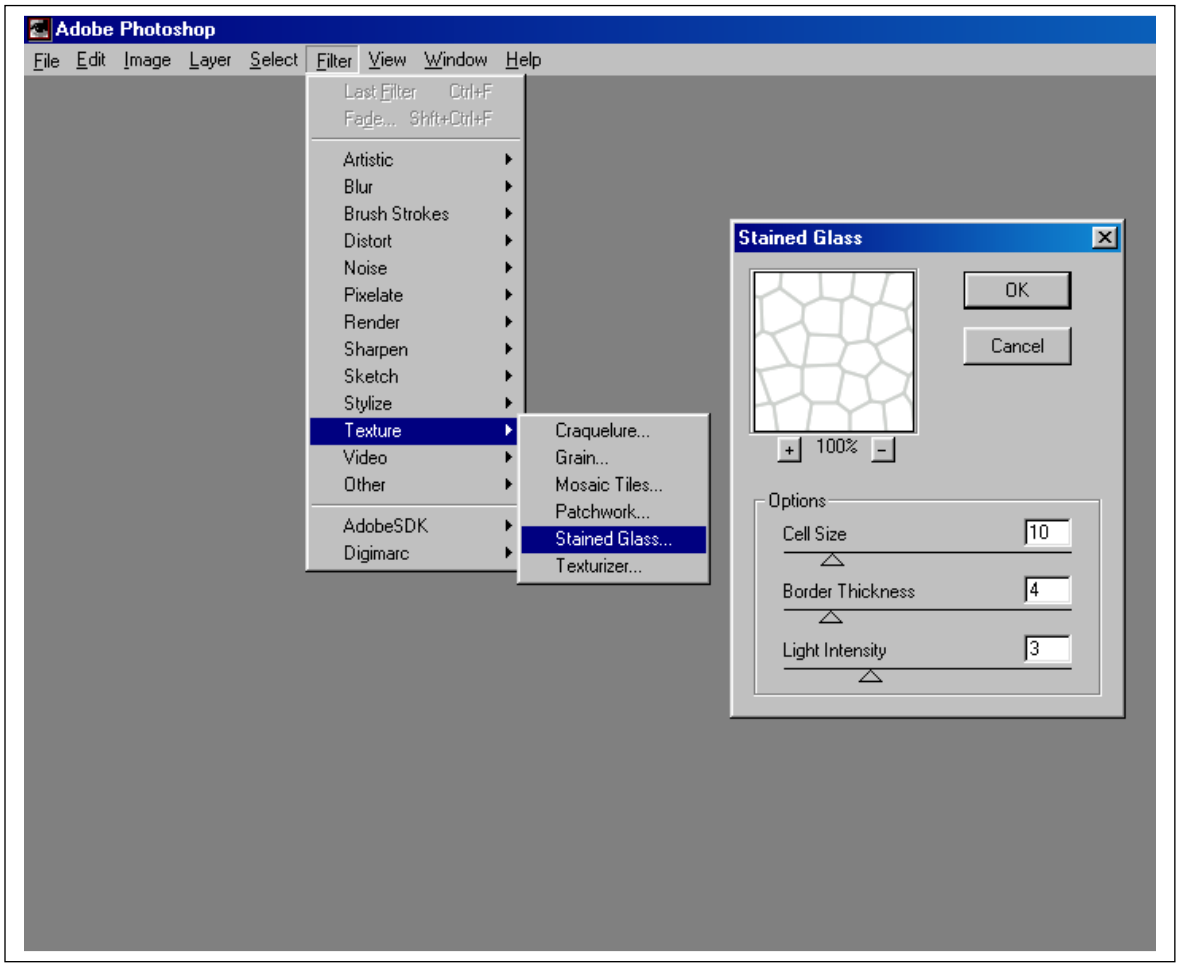

### 3.3.9 FILTRO (FILTER)

Módulos de filtro modificam uma área selecionada de uma imagem existente ou toda a imagem existente. Este módulo está presente no menu "*Filter*". Os *plug-ins* de filtro são os que os usuários do *Photoshop* estão mais familiarizados (figura 7).

Alguns filtros fazem parte do aplicativo *Photoshop*. Outros são módulos externos que se localizam na pasta "*Plug-ins*" da instalação do *Photoshop*. Isso permite que se acrescente uma funcionalidade ao *Photoshop*, adquirindo filtros adicionais de coletâneas de outros fornecedores, por exemplo, o *PhotoTools* (*Extensis*), *Eye Candy* (*Alien Skin*), *Series* (*Andromeda*), *Paint Alchemy* (*Xaos Tools*) e *Kai's Power Tools* (*MetaCreations*).

A implementação deste trabalho demonstrará um *plug-in* de filtro.

**Figura 7 – Filter Plug-in**



## 3.4 EXTENSÕES DOS PLUG-INS

Os arquivos de *plug–in* devem seguir as regras da tabela 01 para serem identificados no *Mac OS* e no *Windows* ([ADO2000]).

**Tabela 01 – Extensões dos plug-ins ( Macintosh e Windows )**

| Tipo de Plug-in | Macintosh File Type | Windows File Extension |
|---|---|---|
| General (any type of plug–in) | 8BPI | .8BP |
| Automation | 8LIZ | .8LI |
| Color Picker | 8BCM | .8BC |
| Import | 8BAM | .8BA |
| Export | 8BEM | .8BE |
| Extension | 8BXM | .8BX |
| Filter | 8BFM | .8BF |
| File Format | 8BIF | .8BI |
| Parser | 8BYM | .8BY |
| Selection | 8BSM | .8BS |

Através das extensões dos *plug-ins*, o *Photoshop* identifica que o arquivo é um *plug-in* e após a identificação ele verifica a estrutura do PiPL do arquivo.

# 4 FUNDAMENTOS DE COMPUTAÇÃO GRÁFICA

A CG é a área da ciência da computação que estuda a geração, manipulação e interpretação de imagens por meio de computadores. Esta definição sugere a subdivisão da área em três grandes subáreas ([PER1989]):

a) síntese de imagens: que se ocupa da produção de representações visuais a partir de especificações geométrica e visual de seus componentes;
b) processamento de imagens: que envolve as técnicas de transformação de imagens em que tanto a imagem partida quanto a imagem resultado apresentam-se sob uma representação visual;
c) análise de imagens: busca obter a especificação dos componentes de uma imagem a partir de sua representação visual.

A síntese de imagens parte da descrição de objetos tais como segmentos de reta, polígonos, poliedros, esferas etc. e produz uma imagem, atendendo ás especificações, em algum meio que possa, em última instância, ser visualizado.

O processamento de imagens parte de imagens já prontas para serem captadas pelos mais diversos recursos: digitalização de fotos, tomadas de uma camêra de vídeo, entre outros. Estas imagens são então transformadas em outras com uma representação visualizável onde características visuais são alteradas.

A análise de imagens desenvolve técnicas para efetuar a operação inversa de síntese de imagens. A partir de uma imagem obtida, por exemplo, por um equipamento de vídeo, procura identificar os elementos que a compõem e determinar suas especificações.

## 4.1 FORMATO DE IMAGENS

Imagens digitalizadas são úteis quando estão armazenadas em uma forma que possa ser utilizada por outras aplicações. Então é necessário colocar estas imagens em uma forma padronizada para que elas possam ser manipuladas por um grande número de aplicações. Programas gráficos podem ser classificados pela forma com a

qual armazenam ou apresentam as imagens. Para esta abordagem há duas categorias: formato de varredura e formato vetorial. O formato varredura (também conhecido como *raster*) é composto por uma série de elementos de imagens, ou *pixels*, que cobrem uma área apresentada, já o formato vetorial envolve o uso de segmentos de linha orientados ao invés de *pixels* para representar uma imagem ([FAL1993]).

### 4.1.1 ARQUIVOS RASTER

Um *raster* é um grupo de amostras discretas em um espaço. Dois critérios determinam os principais tipos de dados que devem ser armazenados em um *raster*. A localização das amostras e o tamanho das amostras. Podem ser consideradas amostras a intervalos uniformes ou a intervalos não uniformes. As amostras podem compartilhar o espaço de forma igual ou desigual ([BRO1996]).

Algumas condições comuns relacionaram a tipo de dados *raster* são definidos conforme:

a) *Raster Device* - qualquer dispositivo que produz um quadro exibindo uma ordem de amostras. Alguns exemplos incluem, televisão, impressoras matriciais, impressoras jato de tinta e impressoras laser.
b) *Pixel* - uma única amostra de um raster 2D de dados. *Pixel* é um abreviação para elemento de quadro. Outro nome equivalente para pixel é *pel*;
c) *Voxel* - uma única amostra de um raster 3D de dados. *Voxel* é um abreviação para elemento de volume;
*d)* *Bit Map* - memória para armazenar imagens raster 2D. Um *Bit map* normalmente se refere a um grupo de *pixels* que é representado por um único *bit* (dígito binário) de memória. Isto restringe cada *pixel* para dois único valores de cor;
e) *Pixel Map* - memória usada para armazenar imagens raster 2D.

## 4.2 TRANSFORMAÇÕES EM 3D

A capacidade para representar e visualizar um objeto em três dimensões é fundamental para a percepção de sua forma. Porém, em muitas situações é necessário mais do que isto, ou seja, poder "manusear" o objeto, movimentando-o através de

rotações, translações e mesmo escala. Assim, as Transformações geométricas em 3D serão representadas por matrizes 4x4 também em coordenadas homogêneas. Dessa forma, um ponto P de coordenadas (x,y,z) será representado por (x,y,z,W). Padronizando (ou homogeneizando) o ponto, tem-se se W for diferente de 0, (x/W, y/W, z/W, 1) ([TAI2000],[FOL1982]).

O sistema de coordenadas para 3D utilizado será o da Regra da Mão Direita. O *OpenGL* utiliza este sistema de coordenada, com o eixo Z perpendicular ao papel e saindo em direção ao observador, como pode ser visto na figura 8. O sentido positivo de uma rotação é dado observando-se sobre um eixo positivo em direção à origem, uma rotação de 90° irá levar um eixo positivo em outro positivo conforme a tabela 2.

**Figura 8 – Sistema de Coodernadas dado pela Regra da Mão Direita**

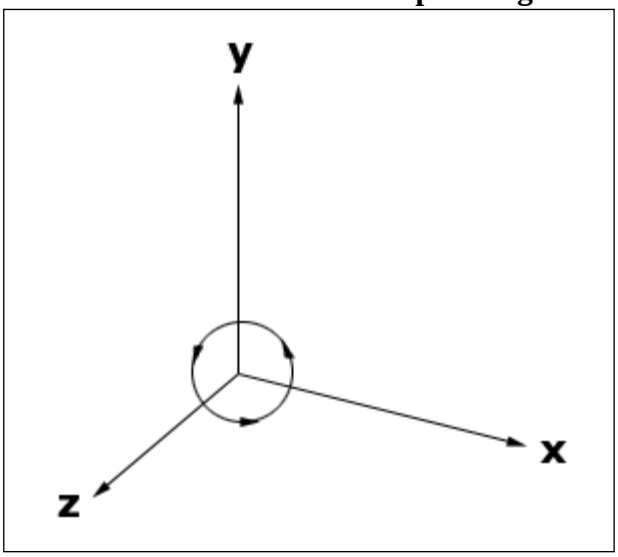



**Tabela 02 – Rotação**

| Eixo de Rotação | Direção da Rotação Positiva |
|---|---|
| X | y para z |
| Y | z para x |
| Z | x para y |

A TRANSLAÇÃO em 3D pode ser vista como simplesmente uma extensão a partir da de 2D (ver quadro 1).

**Quadro 1 - Equação (1a)**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Assim, a equação (quadro 1) pode ser representada também como apresenta-se no quadro 2.

**Quadro 2 – Fórmula (1b)**

$$P' = T(dx,dy,dz) \cdot P$$

Similarmente a ESCALA em 3D, fica conforme apresentada no quadro 3 e quadro 4.

**Quadro 3 – Equação (2a)**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

**Quadro 4 - Fórmula (2b)**

$$P' = S(sx,sy,sz) \cdot P$$

Finalmente, verifica-se como ficam as equações de ROTAÇÃO em 3D, pois as rotações podem ser efetuadas em relação a qualquer um dos três eixos.

A equação de rotação em 2D é justamente uma rotação em torno do eixo z em 3D, a qual é apresentada no quadro 5.

**Quadro 5 – Equação (3)**

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A matriz de rotação em relação ao eixo x é apresentada no quadro 6.

**Quadro 6 – Equação (4)**

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A matriz de rotação em relação ao eixo y apresenta-se no quadro 7.

**Quadro 7 – Equação (5)**

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Para a rotação também temos que P' = R(q) * P.

Os vetores compostos pelas linhas e colunas da submatriz 3x3 do canto superior esquerdo de $R_{x,y,z}$ $(\theta)$ são mutuamente perpendiculares, e a submatriz possui determinante igual a 1, o que significa que as três matrizes são chamadas de Ortogonais Especiais. Além disso, uma sequência arbitrária de rotações é também ortogonal especial. Deve-se lembrar que as transformações ortogonais preservam distâncias e ângulos.

Todas estas matrizes de transformação (T, S e R) possuem inversas. A inversa de T é obtida simplesmente negando-se dx, dy e dz. Para a matriz S, basta substituir sx, sy e sz por seus valores inversos. Para cada uma das matrizes de rotação *R*, basta negar o ângulo de rotação. Além disso, para uma matriz ortogonal *B*, sua inversa ($B^{-1}$) é a sua matriz transposta, ou seja $B^{-1} = B^{T}$.

Uma seqüência arbitrária de transformações de translação, escala e rotação podem ser multiplicadas juntas, resultando a matriz apresentada no quadro 8.

**Quadro 8 – Equação (6)**

$$\mathbf{M} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

A submatriz 3x3 do canto superior esquerdo R, agrega as transformações de escala e rotação, enquanto a última coluna à direita T agrega as translações. Para obter-se um pouco mais de eficiência (ganho computacional) pode-se executar a transformação explicitamente como mostrado no quadro 9.

**Quadro 9 – Equação (7)**

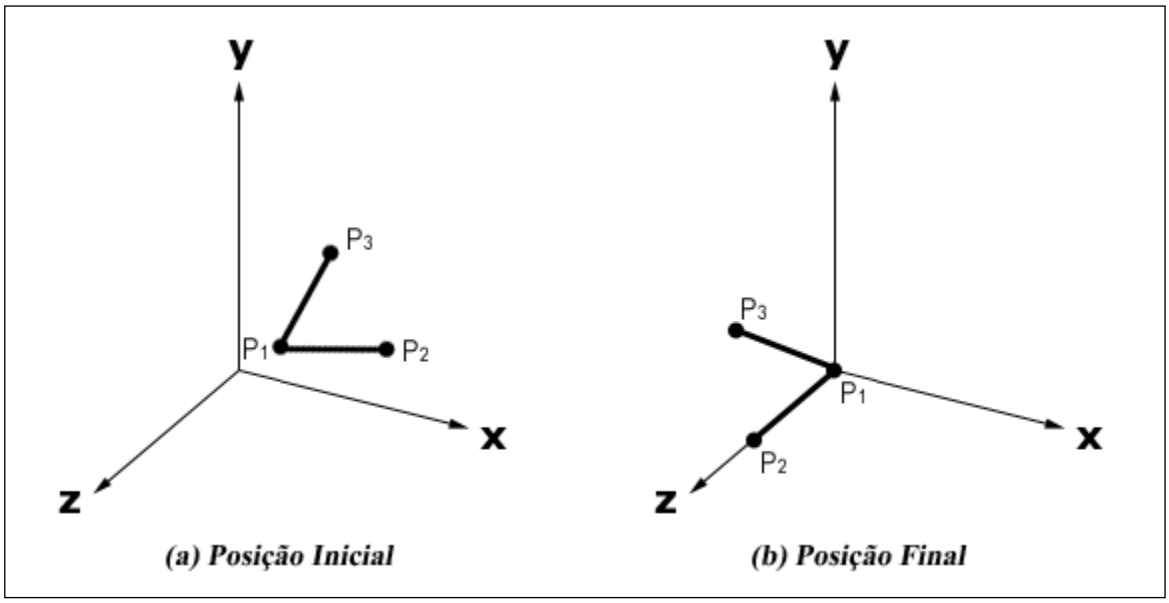
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = R \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + T$$

## 4.3 COMPOSIÇÃO DE TRANSFORMAÇÕES EM 3D

A composição de transformações em 3D pode ser entendida mais facilmente através do exemplo indicado na figura 9. O objetivo é transformar os segmentos de reta $P_1P_2$ e $P_1P_3$ da posição inicial em **(a)** para a posição final em **(b)**. Assim o ponto $P_1$ deve ser transladado para a origem, $P_1P_2$ deverá ficar sobre o eixo **z** positivo, e $P_1P_3$ deverá ficar no plano positivo de **yz**. Além disso, os comprimentos das linhas não devem ser alterados ([TAI2000]).

**Figura 9 – Transformações dos pontos**

Uma primeira maneira de se obter a transformação desejada é através da composição das primitivas de transformação $T$, $R_x$, $R_y$ e $R_z$.



Subdividindo o problema, têm-se os seguintes quatro passos:

a) transladar $P_1$ para a origem;
b) rotacionar o segmento $P_1P_2$ em relação ao eixo **y**, de forma que ele ($P_1P_2$) fique no plano **y-z**.
c) rotacionar o segmento $P_1P_2$ em relação ao eixo **x**, de forma que ele ($P_1P_2$) fique sobre o eixo **z**.
d) rotacionar o segmento $P_1P_3$ em relação ao eixo **z**, de forma que ele ($P_1P_3$) fique no plano **y-z**.

**O primeiro passo**; transladar $P_1$ para a Origem.

A equação de translação é mostrada no quadro 10.

**Quadro 10 – Equação (8)**

Aplicando $T$ a $P_1$, $P_2$ e $P_3$, tem-se:

$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inicialmente, a equação $P_1$ apresenta-se conforme o quadro 11.

**Quadro 11 – Equação (8a)**

$$P_1 = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

A equação $P_2$ apresenta-se conforme o quadro 12.

**Quadro 12 – Equação (8b)**

$$P_2 = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2-x_1 \\ y_2-y_1 \\ z_2-z_1 \\ 1 \end{bmatrix}$$

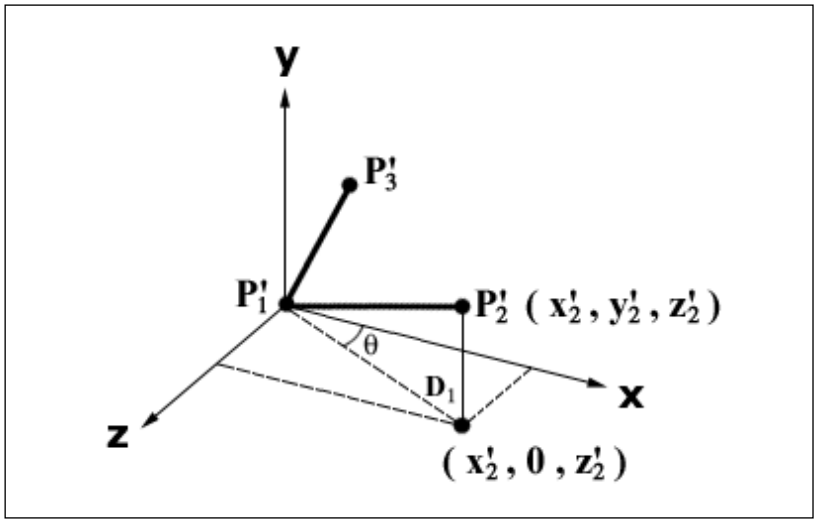
E a equação $P_3$ esta é definida conforme o quadro 13.

**Quadro 13 – Equação (8c)**

$$P_3 = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3-x_1 \\ y_3-y_1 \\ z_3-z_1 \\ 1 \end{bmatrix}$$

**O segundo passo**; rotacionar em relação ao eixo Y é mostrado na figura 10.

**Figura 10 - Rotação no eixo y**




A figura 10 (o ângulo θ indica a direção positiva de rotação em relação a **y**. O ângulo utilizado na realidade é -(90-θ)) mostra $\overline{P_1P_2}$ após o primeiro passo, bem

como a projeção de $\overline{P_1P_2}$ no plano **x-z**. O ângulo de rotação é -(90-θ)=θ-90. Então, o seno e coseno da equação, ficam descritos conforme o quadro 14.

**Quadro 14 – Equação (9)**

$$\sin(\theta - 90) = -\cos\theta = -\frac{X_2}{D_1} = \frac{X_2 - X_1}{D_1},$$

$$\cos(\theta - 90) = \sin\theta = \frac{Z_2}{D_1} = \frac{Z_2 - Z_1}{D_1},$$

Os valores $D_1$, da equação do quadro 14, é resultado da equação do quadro 15.

**Quadro 15 – Equação (10)**

$$D_1 = \sqrt{(Z_2)^2 + (X_2)^2} = \sqrt{(Z_2 - Z_1)^2 + (X_2 - X_1)^2}$$

Então substituindo os valores encontrados, na equação 5 (ver quadro 7), tem-se:

$P_2'' = R_y(\theta\text{-}90) \cdot P_2' = [0\ y_2\text{-}y_1\ D_1\ 1]^T$

Como era esperado o componente *x* de $P_2''$ é zero e *z* possui comprimento $D_1$.

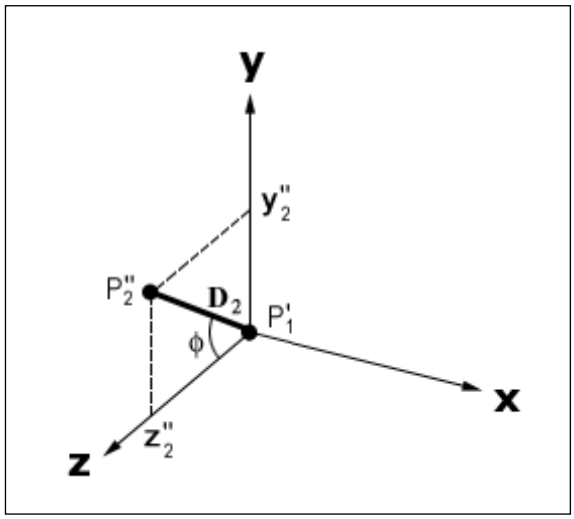
**O terceiro passo**; rotacionar em relação ao eixo X é demonstrado na figura 11.

A figura 11 mostra $\overline{P_1P_2}$ após o segundo passo (o segmento de reta $\overline{P_1P_3}$ não é mostrado porque não é utilizado para determinar os ângulos de rotação. Ambos os segmento são rotacionados por $R_x(\phi)$). O ângulo de rotação é ϕ, e cos(ϕ) e sin(ϕ) , são especificados conforme o quadro 16.

**Quadro 16 – Equação (11)**

$$\cos\phi = \frac{x_2}{D_2} \quad , \quad \sin\phi = \frac{y_2}{D_2}$$

**Figura 11 - Rotação no eixo x**




O valor de $D_2$, especificado na figura 11, é $D_2 = |P_1'' \; P_2''|$ , onde $D_2$ é o comprimento do segmento de reta $\overline{P_1''P_2''}$. Como as translações e rotações preservam o comprimento, então, o comprimento de $\overline{P_1''P_2''}$ é o mesmo de $\overline{P_1P_2}$, a definição de $D_2$ é apresentada no quadro 17.

**Quadro 17 – Equação (12)**

$$D_2 = \left|P_1'' \; P_2''\right| = \left|P_1 \; P_2\right| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

O resultado da rotação no terceiro passo é apresentado no quadro 18.

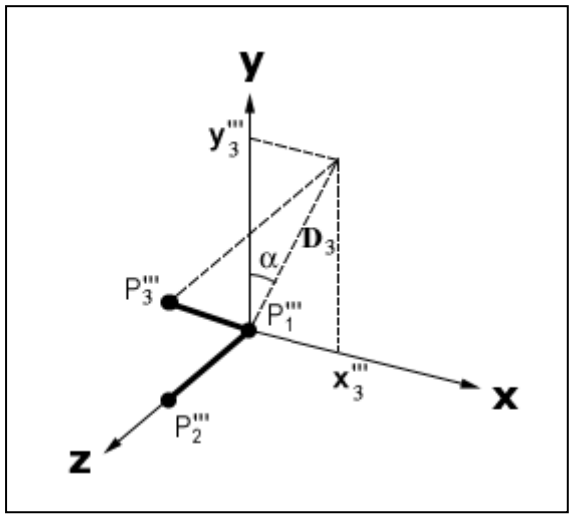
**Quadro 18 – Equação (13)**

$$P_2''' = R_x(\phi) \cdot P_2'' = R_x(\phi) \cdot R_y(\theta - 90) \cdot P_2'$$
$$= R_x(\theta) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = \left[0 \;\; 0 \;\; \left|P_1 \; P_2\right| \;\; 1\right]^T$$

Dessa forma, agora $\overline{P_1P_2}$ está sobre (coincidindo) o eixo **z** positivo.

**O quarto passo**; rotacionar em relação ao eixo Z, demonstrado na figura 12.

**Figura 12 - Rotação no eixo z**




A figura 12 mostra $\overline{P_1P_2}$ e $\overline{P_1P_3}$ após o terceiro passo, com $P_2'''$ sobre o eixo **z** e $P_3'''$ definido conforme o quadro 19.

**Quadro 19 – Equação (14)**

$$P_3''' = \left[ x_3''' \;\; y_3''' \;\; z_3''' \;\; 1 \right] = R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) \cdot P_3$$

A rotação é dada pelo ângulo positivo α, e cos(α) e sin(α) são definidos conforme o quadro 20.

**Quadro 20 – Equação (15)**

$$\cos\alpha = \frac{y_3'''}{D_3} \;\; , \;\; \sin\alpha = \frac{x_3'''}{D_3} \;\; , \;\; D_3 = \sqrt{(x_3''')^2 + (y_3''')^2}$$

$\overline{P_1P_3'}$ é trazido para o plano y-z.

Dessa forma, após o quarto passo o resultado é alcançado como exemplificado na figura 9 (b).

A matriz de composição M, apresentada no quadro 21, é a transformação necessária à composição solicitada na figura 9.

.

**Quadro 21 – Equação (16)**

$$M = R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T$$

Como pode-se ver, a base para transformação e a composição de transformações 3D estão fundamentalmente ligadas a operações em matrizes ([LEM1990]).

## 4.4 TRANSFORMAÇÕES DE VISUALIZAÇÃO

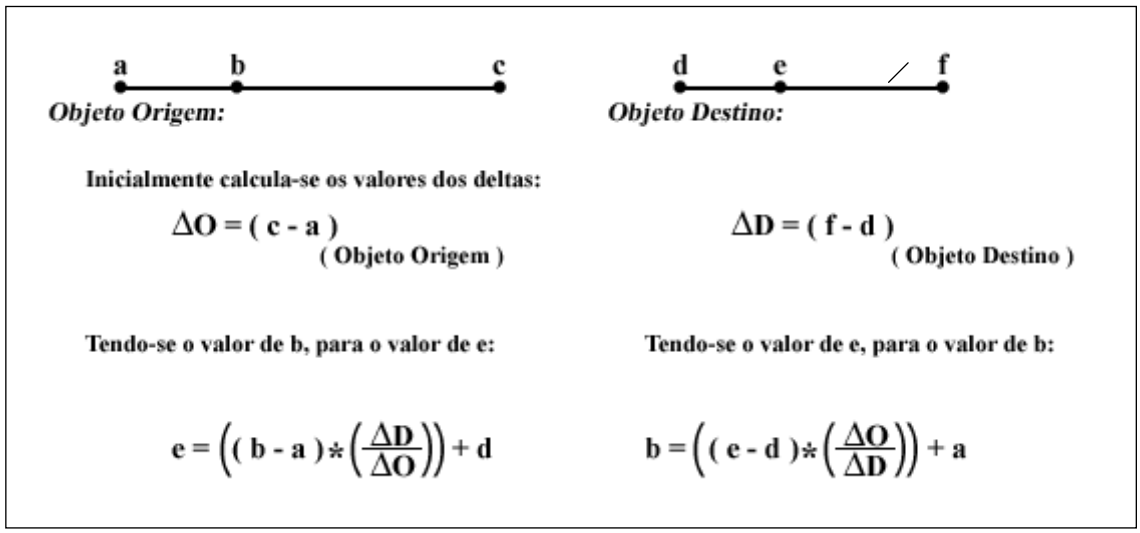
A visualização de uma imagem de um modelo envolve o mapeamento das coordenadas dos pontos e das linhas que constituem o modelo, nas coordenadas apropriadas do dispositivo ou da estação de trabalho, onde a imagem irá ser visualizada. Isto é feito através de transformações de coordenadas conhecidas por transformações de visualização ([ROG1985]).

O sistema de coordenadas do dispositivo físico (*physical device coordinate system*, PDCS) é o sistema de coordenadas correspondente ao dispositivo ou à estação de trabalho onde a imagem do modelo irá ser visualizada.

O sistema de coordenadas do dispositivo normalizado (*normalized device coordinate system*, NDCS) é um sistema de coordenadas direto.

No *Photoshop* o usuário pode definir diferentes dimensões para a imagem, e no *OpenGL* os valores estão atribuídos entre 1 negativo e 1 positivo, tanto para x e y como para z. Sendo assim, faz-se necessário uma conversão dos valores do *Protoshop* para o *OpenGL*. Para esta conversão utiliza-se a fórmula descrita no quadro 22 ([REI2000]).

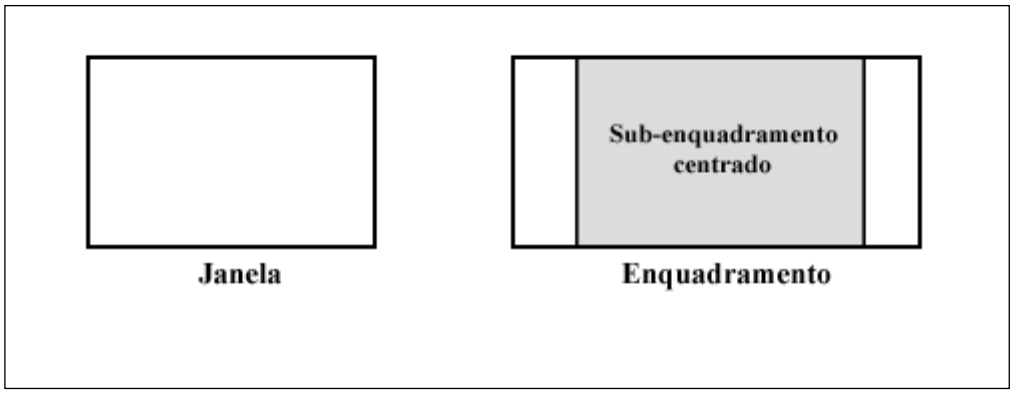
**Quadro 22 – Transformações NDCS**




Uma vez que a transformação de visualização envolve variação de escala, pode acontecer que surjam distorções indesejáveis sempre que ΔD = ΔO. Por exemplo, circunferências dentro de uma janela podem, eventualmente, ser visualizadas como elipses e quadrados como retângulos. Para que estas distorções sejam evitadas, é necessário manter as razões de aspecto (*Aspect Ratio*) iguais entre a origem e o destino. Para isso deve-se fazer com que ΔD seja igual a ΔO ([PLA1991]).

Caso haja diferença entre ΔD e ΔO, pode-se definir uma região dentro do enquadramento do destino, um sub-enquadramento (figura 13).

**Figura 13 – Sub-enquadramento centrado**



# 5 OPENGL

O padrão *OpenGL* permite visualizar modelos tridimensionais complexos construídos a partir de um conjunto de primitivas geométricas simples tais como pontos, linhas e polígonos. Cenas complexas e realistas podem ser criadas utilizando recursos como anti-serrilhado (*antialiasing*), mapeamento de textura, sombra, transparência e outros.

## 5.1 O QUE É OPENGL

*OpenGL* é uma interface de software à hardware de gráficos. Esta interface consiste de cerca de 120 comandos distintos que usa-se para especificar os objetos e operações para produzir aplicações tridimensionais interativas.

*OpenGL* é projetado como uma interface aerodinâmica, hardware-independente a ser implementada em muitas plataformas de hardware diferentes. *OpenGL* não provê comandos de alto-nível para descrever modelos de objetos tridimensionais. Tais comandos podem permitir especificar formas relativamente complicadas como automóveis, partes do corpo, aviões, ou moléculas, para isso, tem-se que construir em cima de seu modelo desejado um conjunto de pequenos modelos geométricos com *OpenGL* - pontos, linhas, e polígonos. Mas em *OpenGL*, ainda não é possível, o uso de efeitos atmosféricos - névoa, nuvem - para demonstrar a presença de partículas no ar; também não é possível o efeito de profundidade-de-campo que simula a inabilidade de uma lente de máquina fotográfica para manter todos os objetos em uma cena fotografada em foco. A máquina fotográfica focaliza em uma parte da cena em particular, e objetos que estão, significativamente, mais distante que aquela seleção ficam um pouco borrados ([SOU2000]).

*OpenGL* é uma biblioteca de rotinas gráficas que permite a criação de aplicações que usam Computação Gráfica em cenas tridimensionais, e apenas envolvem

o domínio de técnicas de visualização de cenas tridimensionais, modelagem e manipulação de objetos sem a preocupação com a matemática envolvida. Além do mais, é possível ter visualização em vários ângulos, com iluminação, mapeamento de textura, *blending* e *antialiasing* (entre outros efeitos) às aplicações criadas.

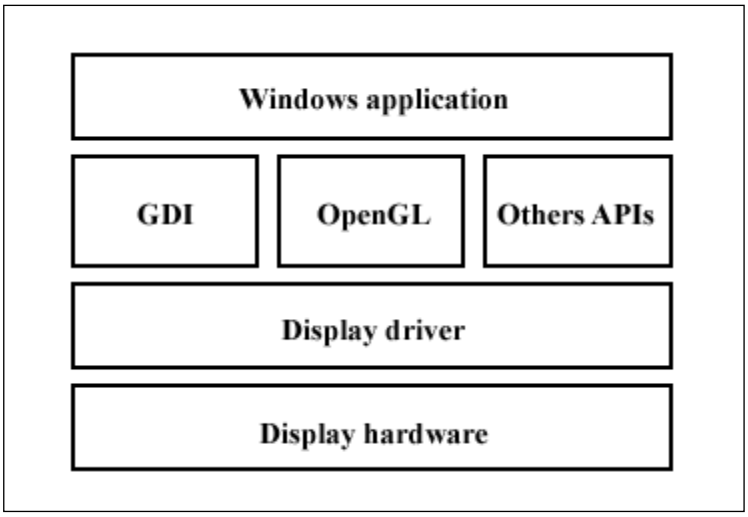
Uma das características mais importantes desta biblioteca é sua implementação em várias plataformas ([SIL2000]):

a) IRIX (Silicon Graphics);
b) AIX (IBM);
c) Windows NT e Windows 95 (PC);
d) Macintosh.

### 5.1.1 O WINDOWS E O OPENGL

A figura 14 demonstra como o *OpenGL* utiliza o driver de exibição que diretamente interage com o hardware. A aplicação *Windows* utiliza o *OpenGL* da mesma maneira que usaria uma *GDI* (Interface Gráfica do Windows) ou outra *API* ([OPE2000]).

**Figura 14 – Relação entre OpenGL e a GDI**




Fonte: [OPE2000]

### 5.1.2 O QUE O OPENGL PODE FAZER

Abaixo estão algumas funções do *OpenGL*:

a) desenhar objetos;
b) visualizar objetos;
c) especificar cores;
d) utilizar iluminação;
e) aplicar efeitos em imagens;
f) manipular imagens;
g) usar texturas;
h) fazer animações;
i) adicionar interatividade.

## 5.2 COMANDOS OPENGL

Apresenta-se nesta seção uma breve descrição da sintaxe e alguns dos principais comandos do *OpenGL*.

### 5.2.1 SINTAXE DOS COMANDOS OPENGL

Os comandos *OpenGL* usam o prefixo *gl* e letras maiúsculas iniciais para cada palavra do comando (`glClearColor`, por exemplo). O *OpenGL* definiu constantes que começam com `GL_`, usam todas as letras maiúsculas, e usam sublinha para separar palavras (como `GL_COLOR_BUFFER_BIT`).

Alguns comandos possuem sufixos (o 3f em `glColor3f`, por exemplo). Em particular, a parte do sufixo, o “3”, indica que três argumentos são determinados. O “f” do sufixo indica que os argumentos são números de ponto-flutuante. Na tabela 3 são demonstrados os sufixos dos comandos *OpenGL* ([OPE2000]).

**Tabela 03 – Sufixos dos Comandos OpenGL**

| Sufixo | Tipo de Dado | Tipo correspondente à linguagem C | Definição do tipo de OpenGL |
|---|---|---|---|
| b | 8-bit integer | signed char | GLbyte |
| s | 16-bit integer | Short | GLshort |
| i | 32-bit integer | Long | GLint, GLsizei |
| f | 32-bit floating-point | Float | GLfloat, Glclampf |
| d | 64-bit floating-point | Double | GLdouble, GLclampd |
| ub | 8-bit unsigned integer | unsigned char | GLubyte, GLboolean |
| us | 16-bit unsigned integer | unsigned short | GLushort |
| ui | 32-bit unsigned integer | unsigned long | GLuint, GLenum, GLbitfield |

### 5.2.2 LIMPANDO UMA JANELA

A memória que guarda o *frame* está normalmente cheia com o último frame desenhado, assim é preciso limpar o *frame* antes de começar a trazer uma nova cena. A cor que utiliza-se para o fundo da janela depende da aplicação.

Como um exemplo, as linhas de código, do quadro 23, limpam a janela para preto.

**Quadro 23 – Limpando a janela**

```
glClearColor (0.0, 0.0, 0.0);

glClear (GL_COLOR_BUFFER_BIT);
```

A primeira linha coloca a cor de fundo para preto, e o próximo comando limpa a janela inteira à cor de fundo atual. O único parâmetro para `glClear()` indica quais *buffers* serão limpos. Tipicamente, fixa-se uma cor de fundo no início de uma aplicação, e então limpa tão freqüentemente quanto necessário os *buffers*.

### 5.2.3 ESPECIFICANDO UMA COR

Para especificar uma cor no *OpenGL*, usa-se o comando `glColor`. O `glColor` possui três parâmetros dos quais são números de ponto flutuante entre 0.0 e 1.0. Os parâmetros são, em ordem, os componentes vermelhos, verdes e azuis da cor (RGB). Por exemplo, o código no quadro 24, especifica a cor vermelha ao máximo, sem verde ou componentes azuis.

**Quadro 24 – Cor vermelha no OpenGL**

```
glColor3f(1.0, 0.0, 0.0);
```

Todos os parâmetros em 0.0 especificam a cor preta; em contraste, todos ao máximo, 1.0, especifica a cor branca. Fixando todos os três componentes a um mesmo nível de cor, ter-se-á a cor cinza. O quadro 25 demonstra algumas cores possíveis com o OpenGL.

**Quadro 25 – Exemplos de cores no OpenGL**

```
glColor3f(0.0, 0.0, 0.0);          preto
glColor3f(1.0, 0.0, 0.0);          vermelho
glColor3f(0.0, 1.0, 0.0);          verde
glColor3f(1.0, 1.0, 0.0);          amarelo
glColor3f(0.0, 0.0, 1.0);          azul
glColor3f(1.0, 0.0, 1.0);          magenta
glColor3f(0.0, 1.0, 1.0);          ciano
glColor3f(1.0, 1.0, 1.0);          branco
```

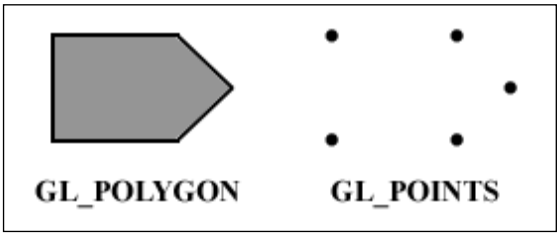
### 5.2.4 DESENHO GEOMÉTRICO DE PRIMITIVAS

Para fazer um desenho no *OpenGL* é necessário especificar qual a posição dos vértices da primitiva geométrica desejada. Para fazer isto, utiliza-se o comando `glVertex`. O comando `glBegin` determina que tipo de primitiva geométrica será construída a partir dos vértices especificados. Por exemplo, o código do quadro 26 especifica os vértices para o polígono mostrado na figura 15.

**Quadro 26 – Definição de uma primitiva geométrica no OpenGL**

```
glBegin(GL_POLYGON);
   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 3.0);
   glVertex2f(3.0, 3.0);
   glVertex2f(4.0, 1.5);
   glVertex2f(3.0, 0.0);
glEnd();
```

**Figura 15 – Exemplo de primitiva geométrica**



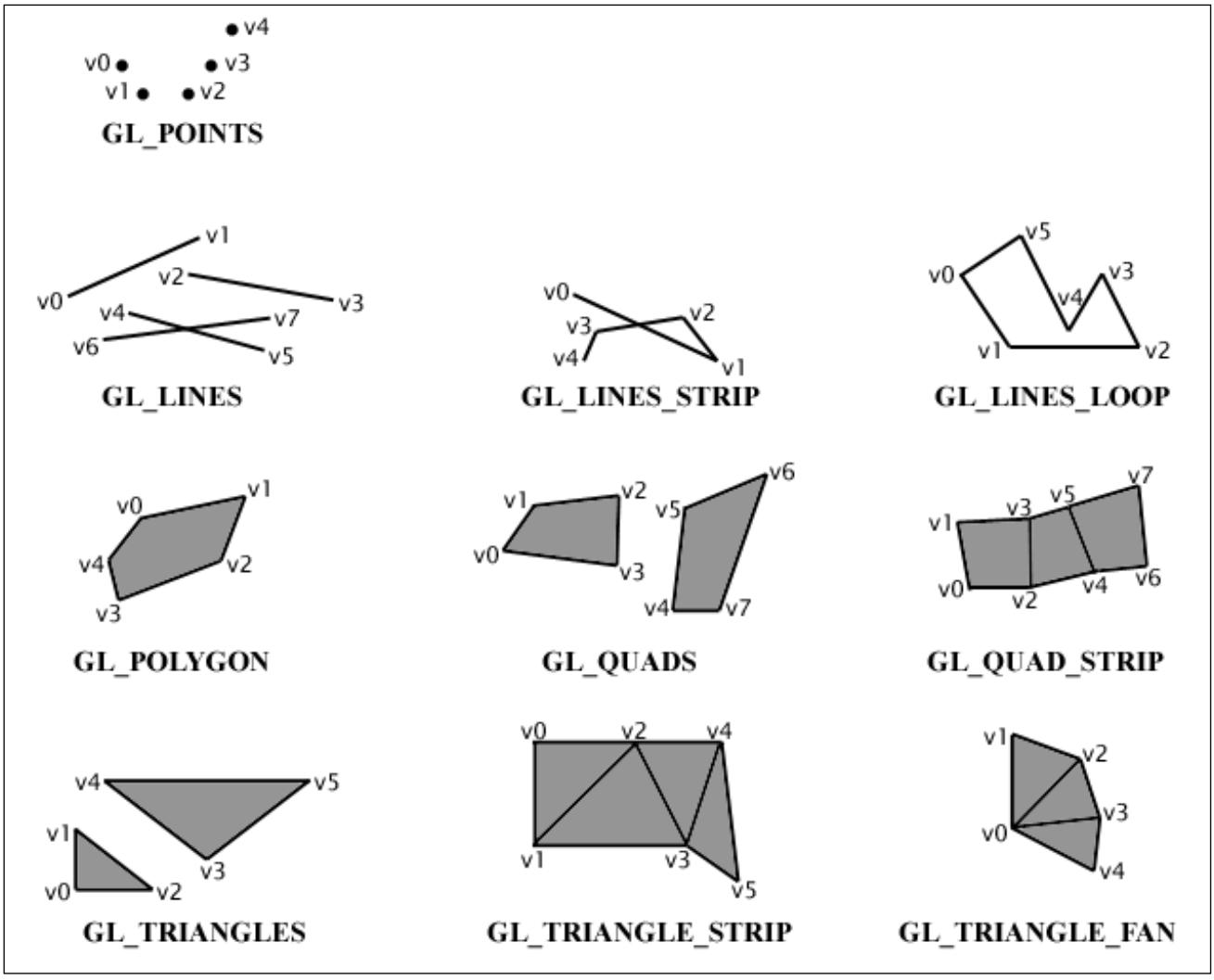

No comando `glBegin`, se fosse utilizado o parâmetro `GL_POINTS` em vez de `GL_POLYGON`, a primitiva geométrica simplesmente teria sido os cinco pontos demonstrados na figura 15. A tabela 4 demonstra um resumo das primitivas atribuídas a função `glBegin()`.

**Tabela 04 – Primitivas Geométricas**

| Valor | Significado |
|---|---|
| GL_POINTS | cada vértice é um único ponto |
| GL_LINES | pares de vértices definem uma linha isolada |
| GL_POLYGON | todos os vértices contidos definem um polígono simples, convexo |
| GL_TRIANGLES | cada três vértices definem um triângulo isolado |
| GL_QUADS | cada quatro vértices definem um quadrilátero isolado |
| GL_LINE_STRIP | os dois primeiros vértices definem uma linha, cada vértice adicional define uma linha que parte do vértice anterior |
| GL_LINE_LOOP | o mesmo que GL_LINE_STRIP, mas após todos os vértices definidos, uma linha é desenhada entre o último e o primeiro vértice |
| GL_TRIANGLE_STRIP | os três primeiros vértices definem um triângulo, e cada vértice adicional define um triângulo a partir dos dois últimos |
| GL_TRIANGLE_FAN | o primeiro vértice é a origem, e cada vértice adicional é combinado com o precedente e a origem para gerar um novo triângulo |
| GL_QUAD_STRIP | os quatro primeiros vértices definem um quadrilátero, e cada par adicional é combinado com o par anterior para gerar um novo quadrilátero |

A figura 16 mostra exemplos de todas as primitivas geométricas descritas na tabela 4.

**Figura 16 – Primitivas Geométricas**



## 5.3 VISUALIZAÇÃO

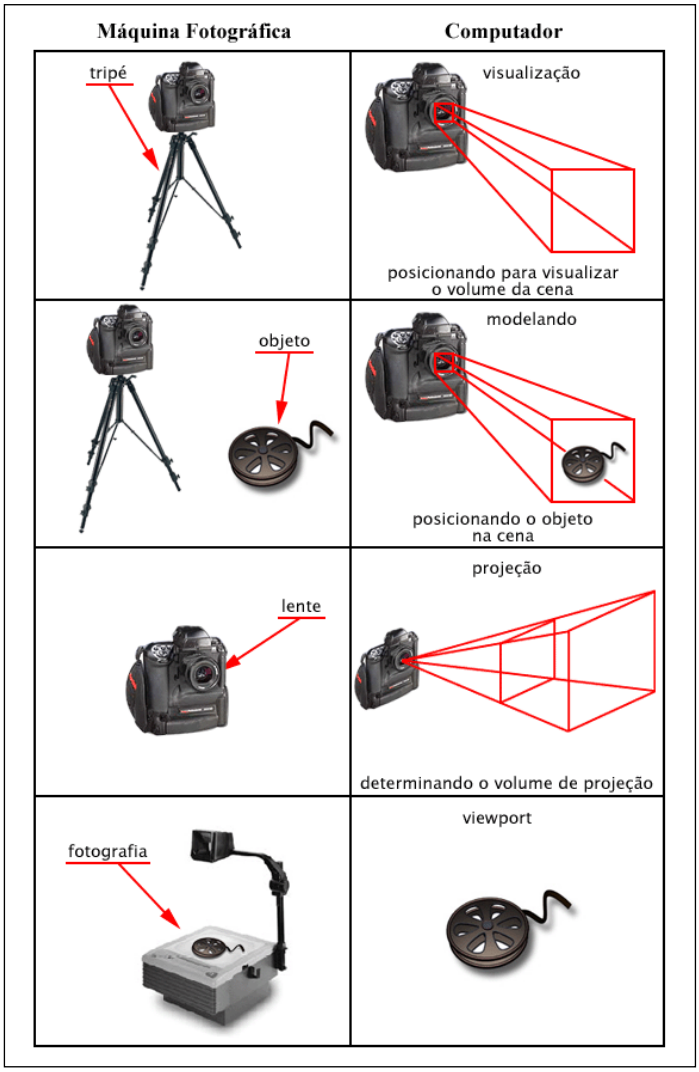
O processo de transformação para produzir uma cena desejada é análogo a fazer uma fotografia com uma máquina fotográfica. Como mostrado na figura 17, os passos com uma máquina fotográfica (ou um computador) poderiam ser os seguintes ([TRE2000]):

a) montando o tripé e apontando a máquina fotográfica à cena (visualizando o objeto);
b) organizando a cena a ser fotografada na composição desejada (modelando o objeto);
c) escolhendo uma lente da máquina fotográfica ou ajustando o zoom (transformação de projeção);
d) determinando o tamanho da imagem desejada na fotografia final (transformação de *viewport*).

Depois que os passos descritos são executados, a cena está pronta.

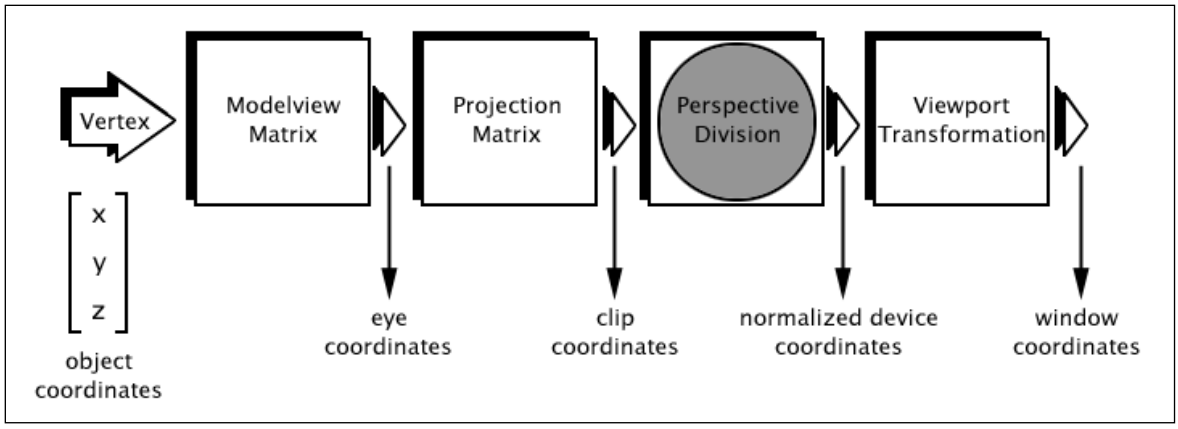
Os passos citados correspondem à ordem na qual se especifica as transformações desejadas no programa, não necessariamente a ordem na qual as operações matemáticas pertinentes são executadas nos vértices de um objeto. A figura 18 demonstra a ordem na qual estas operações acontecem no computador ([TRE2000]).

**Figura 17 – Analogia com uma máquina fotográfica**




Fonte: [TRE2000]

**Figura 18 – Operações no computador**




Fonte: [TRE2000]

## 5.4 TRANSFORMAÇÕES GEOMÉTRICAS

As três rotinas de *OpenGL* para modelar transformações geométricas são `glTranslate`, `glRotate` e `glScale`. Estas rotinas transformam um objeto movendo, girando, aumentando/esticando ou reduzindo (encolhendo o mesmo).

### 5.4.1 TRANSLAÇÃO

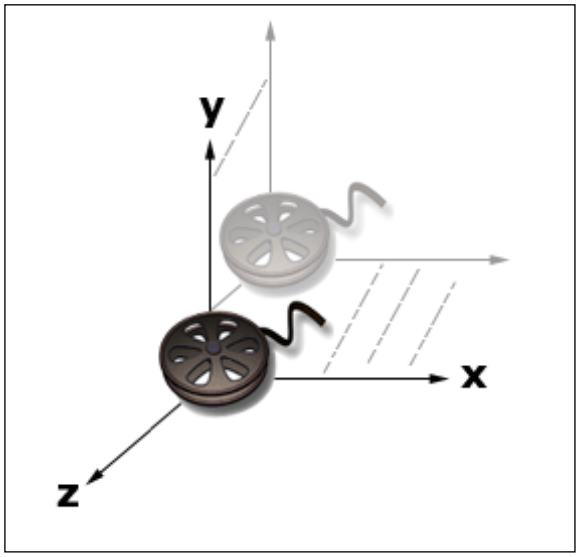
A rotina `glTranslate` (quadro 27), multiplica a matriz atual por uma matriz que mova um objeto por um determinado eixo x, y, ou z.

**Quadro 27 – Rotina glTranslate**

```
void glTranslate{fd} (TYPE x, TYPE y, TYPE z);
```

A figura 19 demonstra o efeito do comando `glTranslate`.

**Figura 19 – glTranslate**



### 5.4.2 ROTAÇÃO

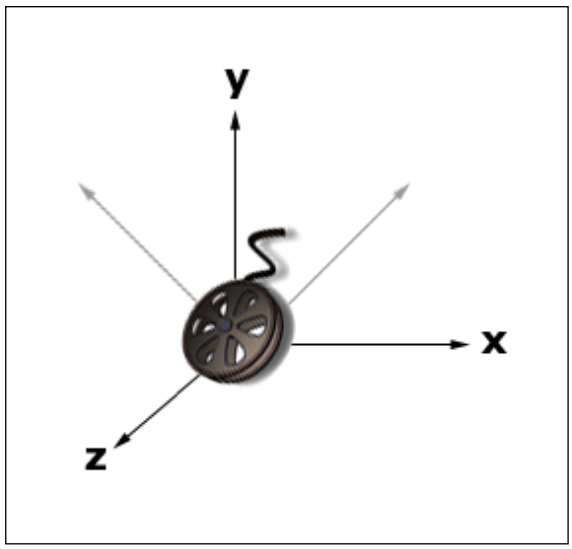
A rotina `glRotate` (quadro 28), multiplica a matriz atual por uma matriz que gira um objeto sobre o raio da origem pelo ponto (x, y, z). O parâmetro de ângulo especifica o ângulo de rotação em graus.

**Quadro 28 – Rotina glRotate**

```
void glRotate {fd} (TYPE ângulo, TYPE x, TYPE y, TYPE z);
```

O efeito de `glRotatef (45.0, 0.0, 0.0, 1.0)` é uma rotação de 45 graus sobre o eixo z. Este exemplo é demonstrado na figura 20.

**Figura 20 – glRotate**




### 5.4.3 ESCALA

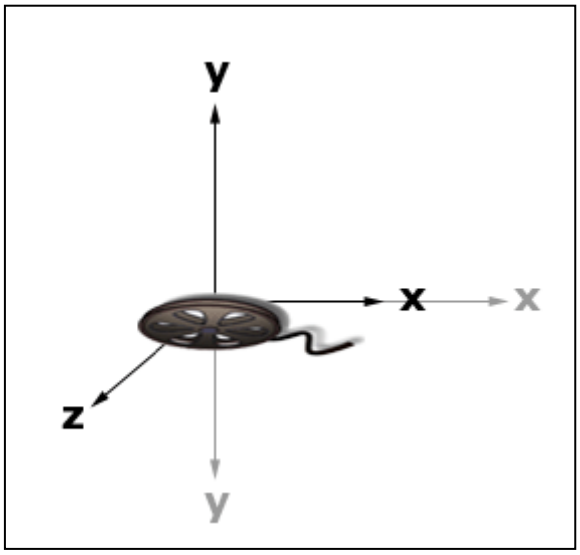
A rotina `glScale` (quadro 29), multiplica a matriz atual por uma matriz que aumenta/estica, diminui/encolhe ou reflete um objeto. Cada coordenada x, y, e z de todo ponto no objeto é multiplicado pelo argumento correspondente x, y, ou z.

**Quadro 29 – Rotina glScale**

```
void glScale{fd}(TYPE x, TYPE y, TYPE z);
```

O efeito A figura 21 demonstra o efeito de `glScalef (2.0,-0.5,1.0).`

**Figura 21 – glScale**



# 6 VISUAL C++

O *Visual C++* é uma ferramenta de programação de 32 *bits*, em modo nativo da *Microsoft*, com um ambiente de desenvolvimento integrado baseado em *Windows* chamado *Developer Studio*, o qual reúne um conjunto de ferramentas como ([CUL1997]):

a) o compilador e as ferramentas para alteração de recursos do *Windows* e geração de código;
b) o *Spy++*, que permite localizar mensagens, classes e informações internas do *Windows*;
c) o *MFC tracer*, que permite o controle de informações relacionadas a programas;
d) o *Pview* (*Process Viewer*), que permite visualizar detalhes sobre fios (*threads*) de execução (ou processos) que estão sendo executados;
e) o *WinDiff*, que mostra a diferença entre dois arquivos;
f) o *InfoViewer* integrado, que forneça acesso on-line a todo o texto dos manuais;
g) *Technical Support*, outro arquivo de ajuda do *Windows* que contém informações da equipe de assistência técnica da *Microsoft* sobre como utilizar o suporte ao produto *Visual C++*;
h) o *Release Notes*, que apresenta as últimas informações sobre o *Visual C++*;
i) o *Help Workshop*, que auxilia na compilação, teste e apresentação de arquivos de ajuda e na edição de arquivos de projeto e de conteúdo;
j) o *MFC Migration Kit*, que você pode usar para mover aplicativos baseados em C para o mundo MFC;
k) o *Microsoft Roadmap*, onde são encontradas informações sobre os produtos, programas e serviços da *Microsoft*.

O *Visual C++* é um produto extremamente rico em recursos, que permite acessar qualquer funcionalidade do *Windows*, mas ao mesmo tempo (ou por causa disto) é um ambiente bastante complexo ([CUL1997]). Além do compilador propriamente dito, a biblioteca de classes MFC, o ambiente integrado de desenvolvimento e todos os utilitários para desenvolvimento no ambiente *Windows* (o que já não é pouco), o *Visual C++* inclui também o *OLE Control Development Kit*. Este kit de desenvolvimento inclui todos os recursos necessários para o desenvolvimento de controles OLE de 32 bits (rebatizados de controles *ActiveX* em função da guerra de marketing pelo controle de mercado de componentes para os browsers da *World Wide Web* na Internet) ([KRU1997]).

Outra peculiaridade do *Visual C++*, decorrente das necessidades de desenvolvimento dentro da própria *Microsoft*, é a existência de um *cross-compiler*[*] para o ambiente *Macintosh*. Este produto permite que código desenvolvido em C++, utilizando a biblioteca de classes MFC, seja compilado de forma a gerar código executável em equipamentos da linha *Macintosh* da *Apple* ([KRU1997]).

## 6.1 PROGRAMAÇÃO BASEADA EM RECURSOS

*Developer Studio* é um ambiente de desenvolvimento integrado (IDE) que é compartilhado pelo *Microsoft Visual C++*, *Microsoft Visual J++*, *Microsoft Visual Basic*, e uma série de outros produtos. O IDE é uma ferramenta de desenvolvimento que está longe do que foi o original *Visual Workbench*, que era baseado no *QuickC for Windows* ([CUL1997]).

Fazem parte do atual *Developer Studio*, *Docking Windows*, barras de ferramentas configuráveis, além de um editor que pode ser personalizado e utilizar macros. A ajuda *online* para esse sistema (*InfoViewer*) trabalha como um *Web browser*.

## 6.2 ARQUIVOS DE PROJETO

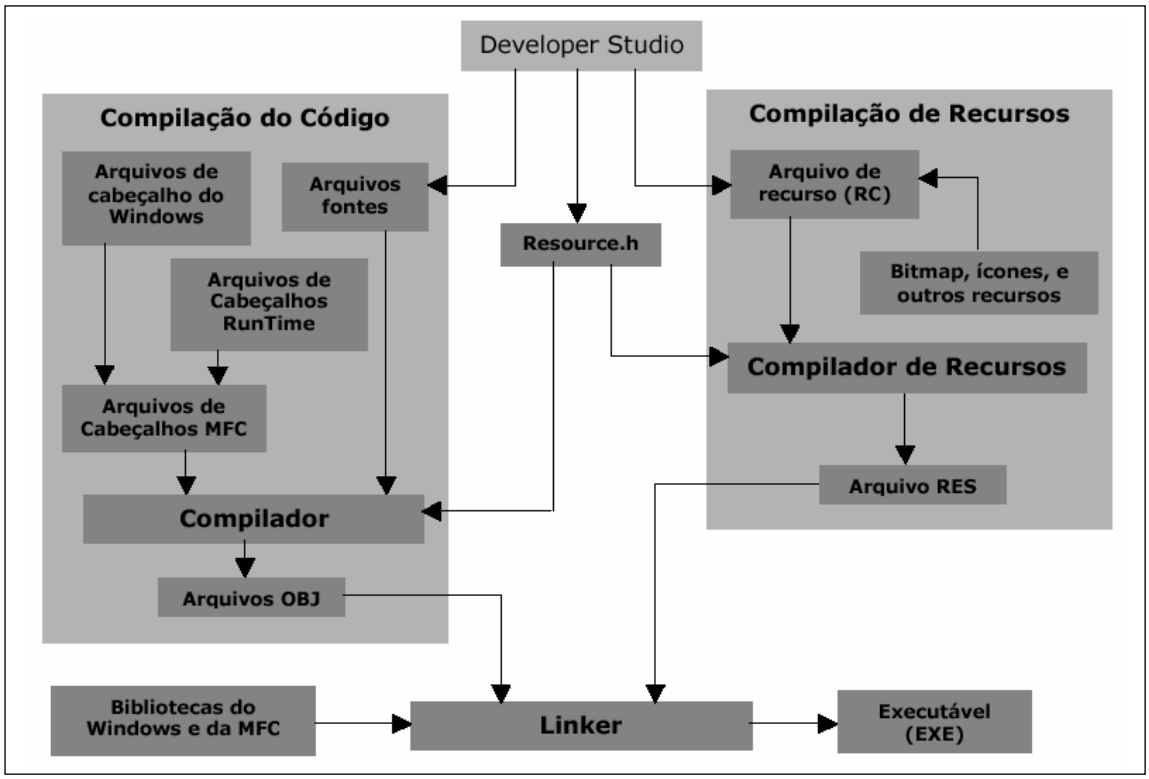
O *Visual C++* organiza os programas em projetos. Um projeto consiste nos arquivos-fontes solicitados por um aplicativo, além das especificações para a estruturação de programa. Cada projeto pode especificar alvos múltiplos para a estruturação a partir de seus arquivos-fonte. Um alvo especifica detalhes como o tipo de

[*] A designação "*cross-compiler*" é dada a um compilador que executa em um determinado sistema (combinação de CPU e sistema operacional), mas gera código a ser executado em outro sistema operacional ou até em uma outra CPU. Por exemplo, as CPUs usadas como eletrônica embarcada,como é o caso de uma CPU que controla a injeção eletrônica de um automóvel, não possuem memória RAM suficiente para compilar seus programas. O desenvolvimento é efetuado em PCs, usando *cross-compilers* que geram código para a CPU embarcada (esse código é normalmente gravado em memórias tipo ROM, já que as CPUs embarcadas na maioria das vezes não possuem unidades de disco).

aplicativo a ser elaborado, a plataforma na qual ele será executado e as definições de ferramentas a serem usadas durante a criação. A inclusão de múltiplos alvos permite que você amplie o escopo de um projeto mantendo ainda um código-fonte coerente a partir do qual pode trabalhar ([CUL1997] [KRU1997]).

Um projeto depende de muitos outros arquivos que não estão em seu subdiretório, tais como arquivos de cabeçalhos (H), arquivos de bibliotecas (LIB). Um *makefile* armazena as opções de compilação e *linker*, bem como expressa o inter-relacionamento dos arquivos fontes. O programa *make* lê esses *makefile* e requisita o compilador, o montador, o compilador de recursos, e *linker* para produzir a saída final, que geralmente é um arquivo executável. O programa *make* usa regras de inferência interna para saber como chamar o compilador para criar um arquivo OBJ a partir de seu código fonte (*.cpp).

**Figura 22 – Processo de construção do Visual C++**




Fonte: [KRU1997]

O *Developer Studio* inclui uma janela de projeto, que exibe o conteúdo do projeto em vários modos de exibição. Como padrão, a janela da área de trabalho do projeto contém os painéis listados na tabela 5.

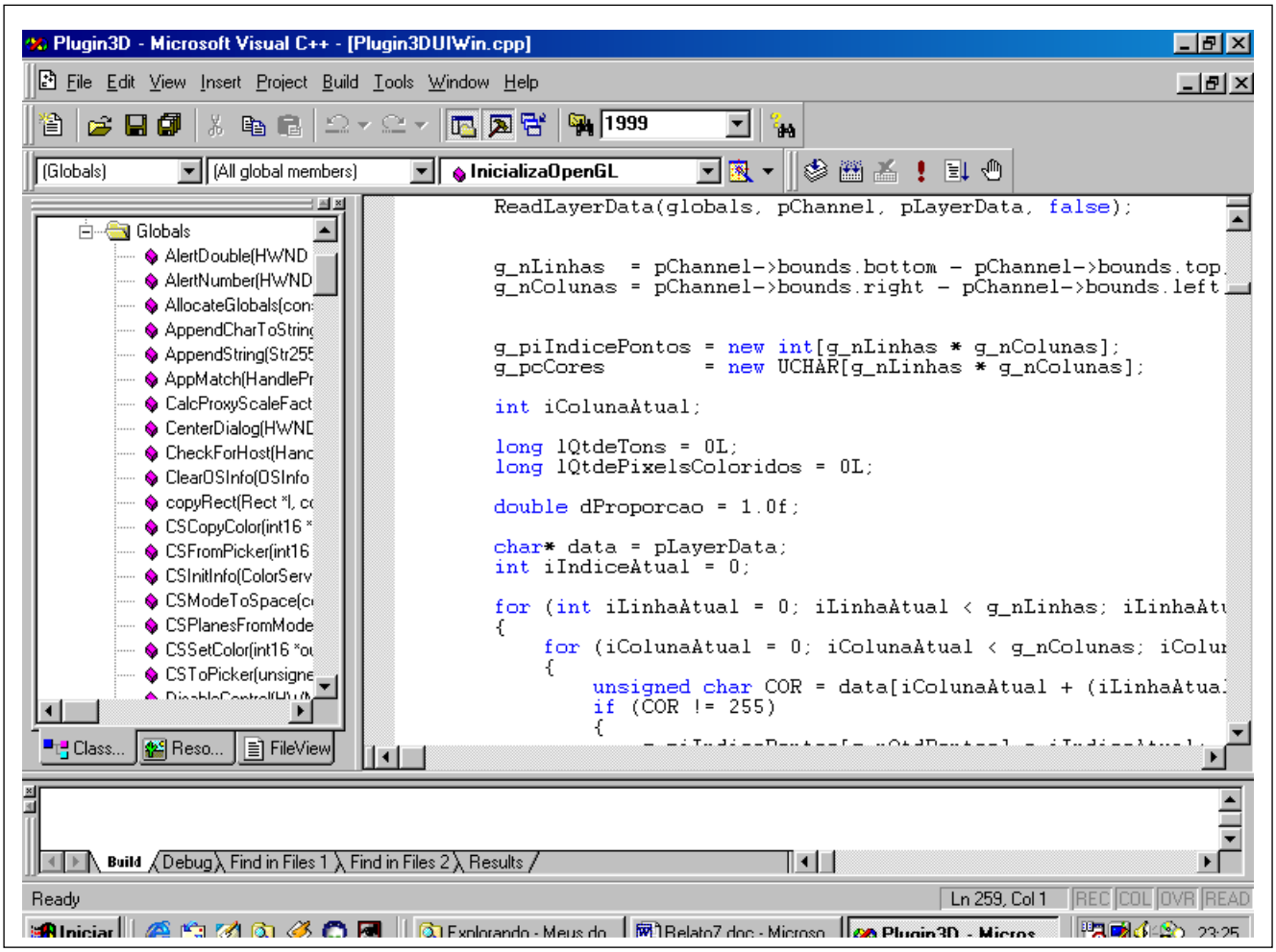
**Tabela 05 – Painéis**

| Nome do Painel | Descrição |
|---|---|
| FileView | Exibe as configurações do projeto criado. |
| ResourceView | Exibe os arquivos de recursos incluídos no projeto. |
| ClassView | Exibe as classes do C++ definidas em seus projetos. |

As áreas de trabalho do projeto são armazenadas em um arquivo com a extensão *.mpd.

Um arquivo para o *workspace* (com a extensão *.dsw) possui uma entrada para cada projeto que compõe este *workspace*. É possível ter vários projetos dentro de um mesmo *workspace*. A figura 23 apresenta a tela do *workspace* do *Visual C++*.

**Figura 23 – Tela do Visual C++**



### 6.2.1 O EDITOR DE RECURSOS – WORKSPACE RESOURCE VIEW

Clicando-se na pasta *ResourceView* na janela *Workspace* do *Visual C++*, é possível selecionar um recurso para a edição. A janela principal é anfitriã de um editor de recurso destinado a cada tipo de recurso selecionado. A janela pode hospedar um editor de menus, um potente editor gráfico para caixas de diálogo, e inclui ferramentas para editar ícones, *bitmaps*, e *strings*. O editor de diálogos permite ainda que sejam adicionados controles *ActiveX*, controles padrões do *Windows* os novos *Windows common controls* ([YAO1992]).

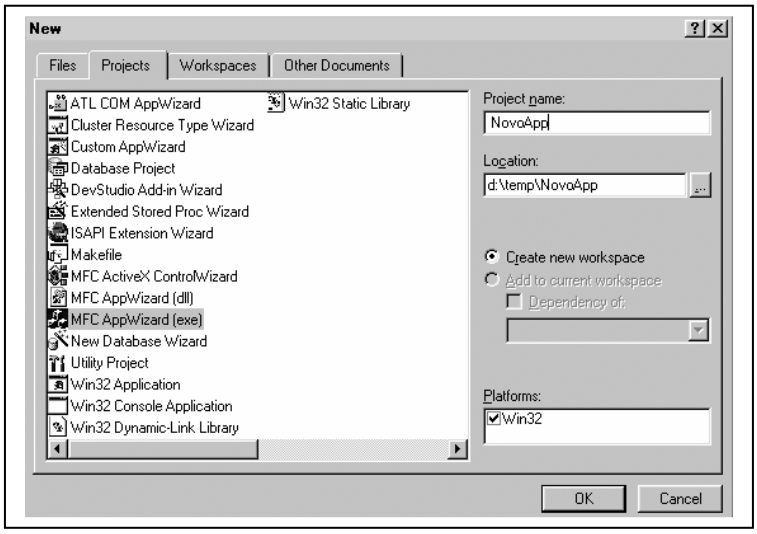
Usualmente, cada projeto contém um arquivo de recurso *script* (*.rc) no formato texto, que descreve os menu, diálogos, string, e aceleradores do projeto. O arquivo com extensão *.rc, possui declarações `#include` para trazer recursos de outros subdiretórios. Estes recursos incluem itens específicos do projeto, como arquivos de *bitmap* (*.bmp) e de ícone (*.ico), e recursos comum a todo programa em *Visual C++* tal como *strings* de mensagem de erro.

### 6.2.2 A CRIAÇÃO DE APLICATIVOS COM O APPWIZARD

O *AppWizard*, um utilitário interno do *Visual C++*, ajuda-nos a iniciar projetos com rapidez através da criação de uma estrutura básica de aplicativo. A estrutura básica é construída com as *Microsoft Foundation Classes* (MFC) e incorpora suporte para recursos, como *Windows Sockets* ou OLE 2.0 ([SWA1994]).

A caixa de diálogo "*New*" aparece, escolha-se *Project Workspace* na lista de opções disponíveis. A caixa de diálogo *New Project Workspace* aparece (figura 24) na caixa de texto "*Name*", digita-se **NovoApp** e então clica-se no botão "*Create*".

**Figura 24 – A caixa de diálogo New Project**




Neste ponto, irá se passar por uma série de passos cada qual fazendo perguntas sobre o aplicativo que deverá ser criado (ver Figura 25). Para aceitar o valor padrão, clica-se no botão "*Next*" através dos seis passos. Em seguida, um clique no botão "*Finish*", a fim de criar o código-fonte para o aplicativo e para o novo arquivo de projeto.

Depois de criar o projeto, pode-se então compilar o novo programa escolhendo-se "*Build*" ( ou pressionado-se Shift + F8). À medida que o projeto é compilado, a janela de saída, na parte inferior da tela, exibe informações de *status* (ver Figura 26). Quando a compilação estiver terminada, escolha-se "*Build*", "*Execute*" (ou pressiona-se Ctrl+F5).

**Figura 25 – Passo do MFC AppWizard**

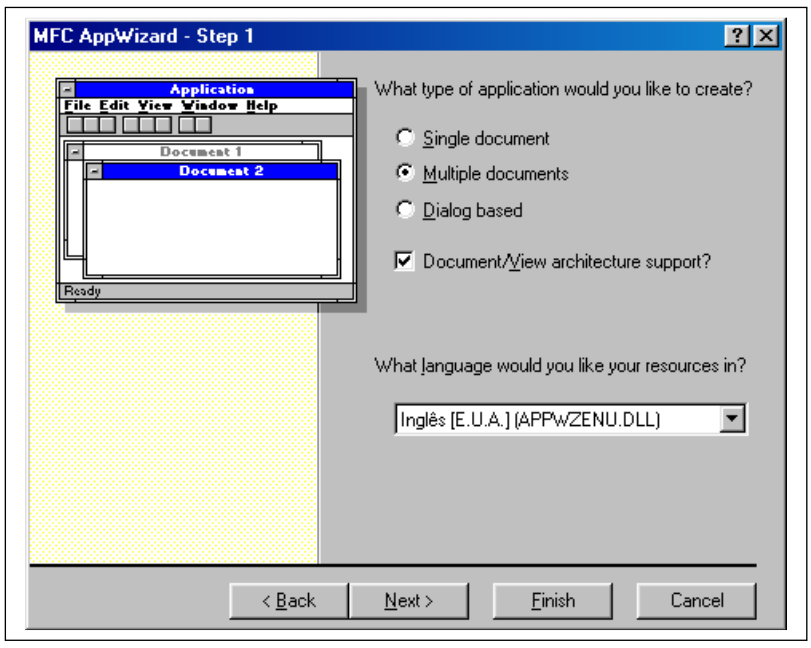
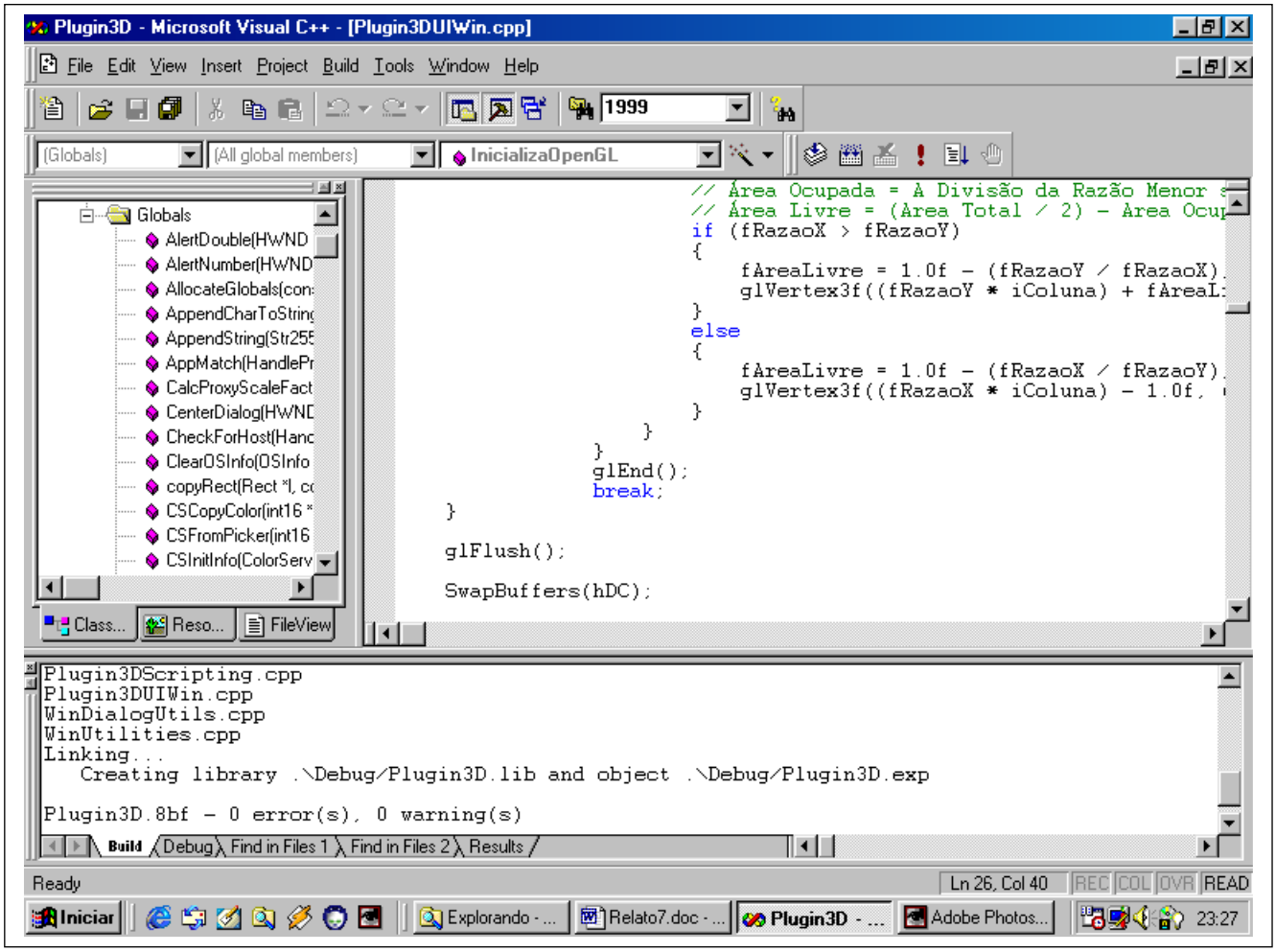


**Figura 26 – A janela de saída da compilação**



# 7 DESENVOLVIMENTO

O protótipo foi desenvolvido na linguagem de programação *Visual C++ 6.0* da *Microsoft* com a incorporação da biblioteca *OpenGL*. O padrão *OpenGL* para vizualização dos dados é baseado no sistema de coordenadas dado pela regra da Mão-Direita descritos no capítulo 4.

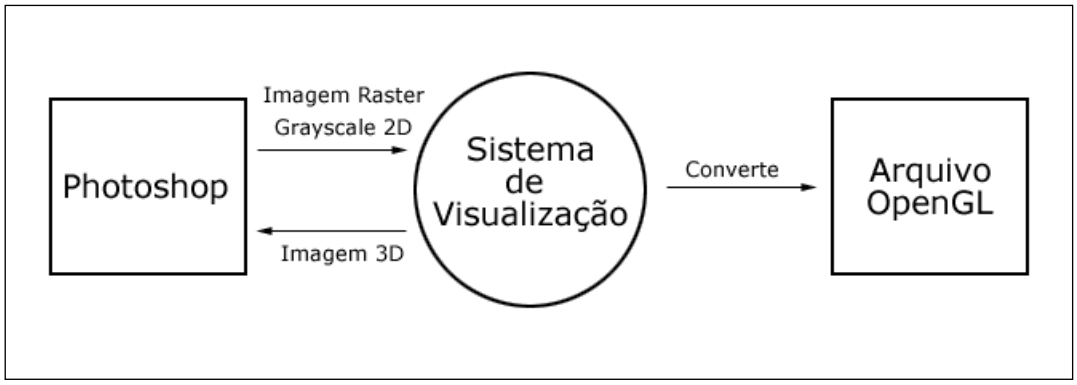
## 7.1 ESPECIFICAÇÃO

Apresenta-se na especificação uma relação dos algoritmos mais utilizados para a conversão das imagens 2D para 3D, bem como a estrutura dos identificadores dos arquivos de *plug-ins* para o *Photoshop*.

### 7.1.1 DIAGRAMA DE CONTEXTO

O diagrama de contexto do protótipo é apresentado na figura 27.

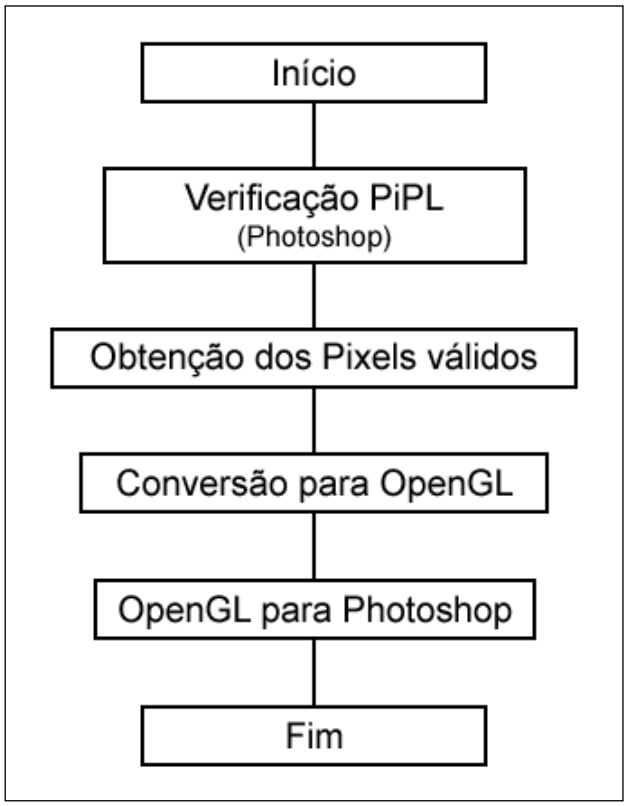
**Figura 27 - Diagrama de contexto**



### 7.1.2 FLUXOGRAMA DO PROTÓTIPO

A ordem de execução das funções do protótipo é apresentada na figura 28.

**Figura 28 – Fluxograma do protótipo**




### 7.1.3 ADOBE PHOTOSHOP PIPL

A PiPL (*Plug-in Property List*) é uma lista que contêm todas as informações necessárias para que o *Photoshop* possa identificar e carregar os módulos de *plug-ins*, como também *flags* e outras propriedades que controlam a operação de cada *plug-in*.

As PiPLs são utilizadas por vários aplicativos da *Adobe*, entre eles o *Adobe After Effects*, o *Adobe Illustrator*, o *Adobe PageMaker*, o *Adobe Premiere*, o *Adobe PhotoDeluxe* e o *Adobe Photoshop*. Quando o *Photoshop* é inicializado, ele procura dentro do diretório dos *plug-ins* todos PiPLs válidos ([ADO2000]).

A estrutura do PiPL (ver quadro 30) possui um número de versão, um contador que indica a quantidade de propriedades contidas na estrutura e uma variável que indica o comprimento da estrutura de dados das propriedades.

**Quadro 30 – Lista de propriedades do PiPL**

```
typedef struct PIPropertyList
{
    int32               version;    /* Versão Corrente = 0
    int32               count;      /* 0 = Nenhuma Propriedade
    PIProperty          properties[1];
} PIPropertyList;
```

Cada propriedade contém um código do fornecedor (todos as propriedades dos plug-ins do *Photoshop* possuem o código do fornecedor igual a ’8BIM’), uma chave (ver Tabela 6) , um identificador (reservado para uso futuro, o valor padrão é zero) e uma váriavel que possui o tamanho do campo da propriedade (ver quadro 31).

**Quadro 31 - Estrutura do PiPL**

```
typedef struct PIProperty
{
    OSType              vendorID;
    OSType              propertyKey;
    int32               propertyID;
    int32               propertyLength;
    char                propertyData[1];
} PIProperty;
```

A tabela 6 apresenta uma relação de alguns tipos de propriedades aceitas pela estrutura do PiPL. O nome da propriedade é definida pela chave setada no PiPL.

**Tabela 06 – Propriedades da estrutura PiPL**

| Tipo | Nome | Chave | Descrição |
|---|---|---|---|
| OSType | PIKindProperty | 'kind' | Especifica qual o tipo do plug-in criado. O protótipo proposto refere-se a um *plug-in* de filtro. '8BFM'=Filter module |
| int32 | PIVersionProperty | 'vers' | 32 bits |
| int16 | PIPriorityProperty | 'prty' | Utilizado para definir a ordem de carga do *plug-in* no Photoshop. |
| CString | EnableInfo | 'enbl' | Define quando o plug-in estará habilitado para ser utilizado, através do modo de imagem. Alguns tipos aceitos: GrayScaleMode, IndexedMode, RGBMode, CMYKMode entre outros. |
| PString | PICategoryProperty | 'catg' | Especifica qual o nome do sub-menu que será colocado no menu **Filter**. |
| PString | PINameProperty | 'name' | Nome do plug-in que será colocado no sub-menu especificado na propriedade PICategoryProperty. |
| PString | PIWin32x86CodeProperty | 'wx86' | Especifica para qual estrutura irá ser colocado os valores da imagem do Photoshop |

### 7.1.4 DEFININDO A ÁREA E AS CORES UTILIZADAS

Como foi citado no capítulo 4, o *Photoshop* trabalha com diferentes tamanhos de imagens e o *OpenGL* possui uma área especificada entre –1 a 1, fazendo-se necessária à conversão dos pontos x,y do *Photoshop* para o *OpenGL*.

Para a definição da área e das cores válidas (cores diferentes de 255 – branco), foram criadas duas listas contendo a posição e a cor do pixel na imagem do *Photoshop* para aumentar a velocidade do mapeamento da imagem no *OpenGL* (ver

quadro 32). O *Photoshop* disponibiliza uma lista contendo todas as cores dos pixels utilizados.

**Quadro 32 – Obtenção dos pixels válidos**

```
IndiceAtual = 0;
QtdPontos = 0;
for (LinhaAtual=0;LinhaAtual<LinhasPhotoshop;LinhaAtual++)
{
  for (ColunaAtual=0;ColunaAtual<ColunasPhotoshop;ColunaAtual++)
  {
    Cor=DadosPhotoshop[ColunaAtual + (LinhaAtual * ColunasPhotoshop)]
    If (Cor != 255)
    {
      IndicePontos[QtdPontos] = IndiceAtual;
      CoresValidas[QtdPontos] = Cor;
      QtdPontos++;
    }
    IndiceAtual++;
  }
}
```

Após a obtenção dos dados válidos, faz-se necessário a conversão dos pontos para o *OpenGL* (ver quadro 33 e quadro 34). Neste exemplo está demonstrado a conversão dos *pixels* válidos para a geração de linhas no *OpenGL*.

**Quadro 33 – Conversão para o OpenGL – NCD (1)**

```
RazaoX = 2 / ColunasPhotoshop;
RazaoY = 2 / LinhasPhotoshop;
for (i=0;i<QtdPontos;i++)
{
  Coluna = IndicePontos[i]%ColunasPhotoshop; /*Pega o Resto da Divisão
  Linha  = IndicePontos[i]/ColunasPhotoshop; /*Linha é um inteiro
  Z      = (((256-CoresValidas[i])/256) * -1);
  glColor3f(CoresValidas/256, CoresValidas/256, CoresValidas/256);
  if RazaoX == RazaoY
  {
    glVertex3f((RazaoX * Coluna) – 1,((RazaoY * Linha) – 1) * -1, 0);
    glVertex3f((RazaoX * Coluna) – 1,((RazaoY * Linha) – 1) * -1, Z);
  }
  ...
```

**Quadro 34 – Conversão para o OpenGL – NCD (2)**

```
...
  else
  {
    if (RazaoX > RazaoY)
    {
    AreaLivre = 1 – (RazaoY / RazaoX)
    glVertex3f((RazaoY * Coluna) + AreaLivre – 1,((RazaoY * Linha) – 1) * -1, 0);
    glVertex3f((RazaoY * Coluna) + AreaLivre – 1,((RazaoY * Linha) – 1) * -1, Z);
    }
    else
    {
    AreaLivre = 1 – (RazaoX / RazaoY)
    glVertex3f((RazaoX * Coluna) – 1,((RazaoX * Linha) + AreaLivre – 1) * -1, 0);
    glVertex3f((RazaoX * Coluna) – 1,((RazaoX * Linha) + AreaLivre – 1) * -1, Z);
    }
  }
}
```

Pode-se observar que a razão de aspecto entre o *Photoshop* e o *OpenGL* continua a mesma após a conversão, proporcionando uma visualização mais próxima da original.

Após a confirmação pelo usuário (botão OK) no protótipo, faz-se necessária a conversão da imagem do *plug-in* para o *Photoshop*; para isso, transforma-se a imagem do *OpenGL* para um *bitmap* e após é repassado essa imagem para o *Photoshop* através do algoritmo apresentado no quadro 35.

**Quadro 35 – Conversão OpenGL (Bitmap) para Photoshop**

```
short itemWidth = (short)(pChannel->bounds.right - pChannel->bounds.left);
short itemHeight = (short)(pChannel->bounds.bottom - pChannel->bounds.top);
float Razao_x = (float)itemWidth / (float)filterWidth;
float Razao_y = (float)itemHeight / (float)filterHeight;
int Size = itemWidth * itemHeight;
LPBYTE lpBits  = g_clientCapture.m_pBits;
INT* pw32Bits  = (INT*)lpBits;
INT*  vetor_x  = new int[itemWidth];
INT*  vetor_y  = new int[itemHeight];

if (Razao_x = Razao_y)
{
 if ((int)Razao_x > 0)
 {
  for (a = 0; a < itemWidth; a++)
  {
   vetor_x[(int) a] = 1;
   vetor_y[(int) a] = 1;
  }
  a = 0;
  for (qtd = 0; qtd < filterWidth; qtd++)
  {
   vetor_x[(int) a] = 0;
   vetor_y[(int) a] = 0;
   a = a + Razao_x;
  }
 }
}

for (y = 0; y < itemHeight; y++)
{
 int i = y * itemWidth;
 if (vetor_y[y] != 0)
 {
  if (vetor_y[y] == 1)  //copia linha anterior
  for (int a = itemWidth; a > 0; a--)
   data[(Size - (y * itemWidth)) - a] = data[(Size - ((y - 1) * itemWidth)) - a];
 }
 else {
  for (x = (itemWidth - 1); x > -1; x--) {
   if (vetor_x[((x + 1) - itemWidth) * -1] == 0) {
     b = (BYTE)(*pw32Bits);
     pw32Bits++;
   }
   int nGray = b;
   data[((Size - i) - x) - 1] = nGray;
  }
 }
}
```

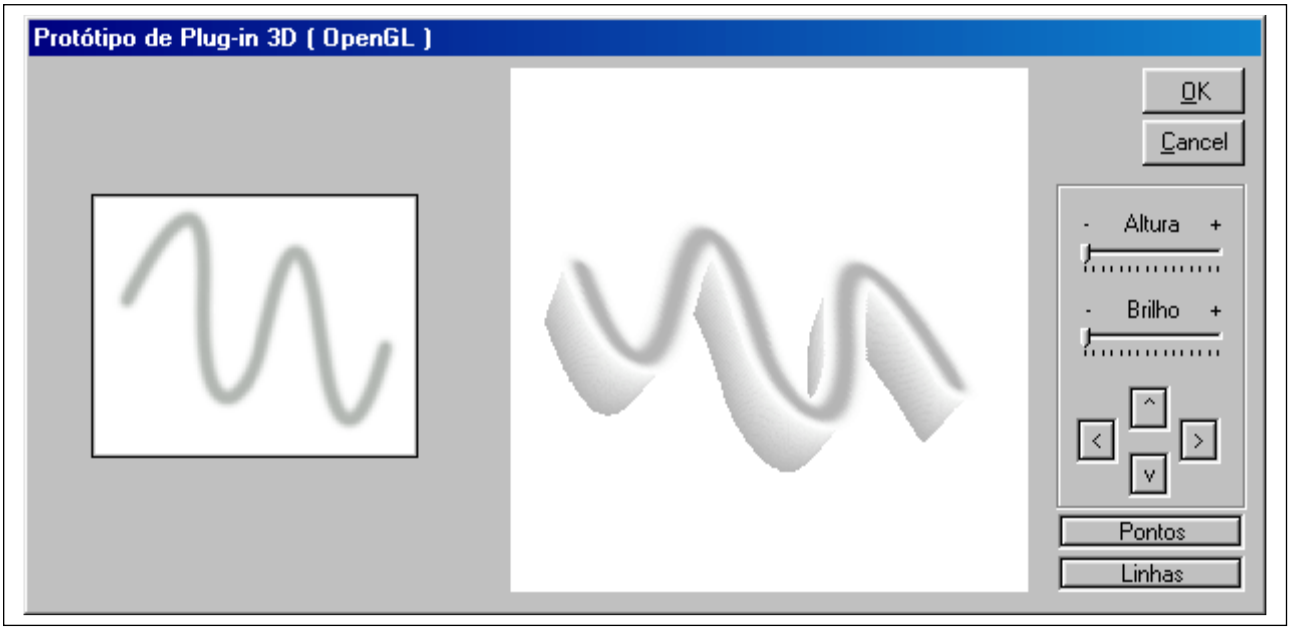
## 7.2 SISTEMA

O “*Plug-in 3D*” foi implementado para trabalhar somente sobre imagens em *grayscale*; se a imagem estiver em outro formato, será necessário convertê-la para após aplicar o efeito do *plug-in*. O *Photoshop* possui *plug-ins* para a conversão de formatos de imagens.

### 7.2.1 FUNÇÕES BÁSICAS DO SISTEMA

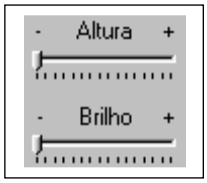
A janela do sistema possui botões que permitem a interação do usuário com o *plug-in*. Na figura 29, podemos ver a janela do sistema.

**Figura 29 – Janela do sistema (visualização em linhas)**



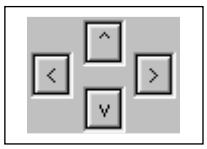
A janela demonstra o objeto inicial (lado esquerdo da janela) e a imagem gerada pelo *OpenGL* (centro da janela). Têm-se no sistema botões *sliders* de “Altura” e “Brilho” (figura 30), que permitem manipular a altura (eixo z) e o brilho (tonalidade da cor) da imagem.

**Figura 30 – Botões *Sliders***




Os botões de Posição (figura 31) permitem alterar os valores do eixo x e y, permitindo a movimentação da imagem.

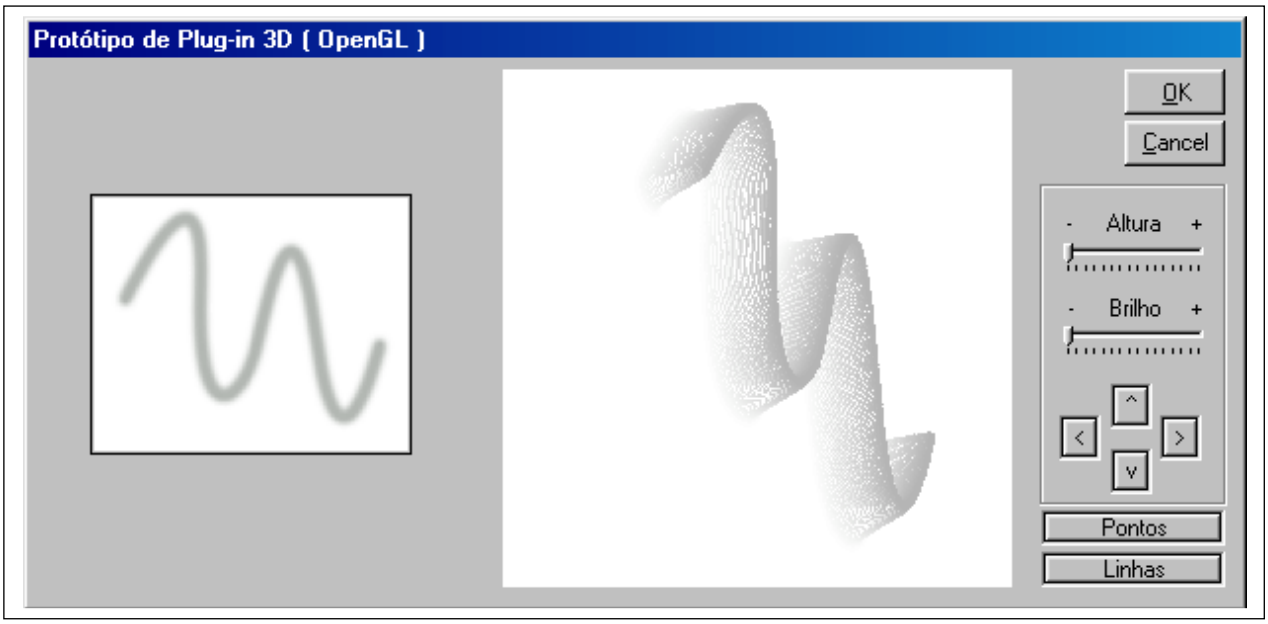
**Figura 31 – Botões de posição**



Clicando-se com o botão esquerdo do mouse e mantendo pressionado o mesmo, pode-se inspecionar (rotacionar) o objeto em diversos ângulos.

Os botões de “Pontos” e “Linhas” alteram o tipo de formação da imagem. No botão “Pontos” a imagem será representada apenas por pontos. A posição do ponto é especificada pelos eixos x e y da imagem inicial, o eixo z é especificado pela tonalidade de cinza presente na imagem inicial. No botão “Linhas” o vértice inicial é especificado pelos pontos x e y da imagem inicial e para o eixo z é especificado o valor zero, o vértice final é encontrado através da perpendicular do vértice inicial, tendo o eixo z alterado para a posição obtida pela tonalidade de cinza presente na imagem inicial. Pode-se observar os resultados desses botões nas figuras 29 (Linhas) e 32 (Pontos).

**Figura 32 – Janela do sistema (visualização em pontos)**




O *plug-in* permite ainda a interação através do *mouse*, clicando-se com o botão esquerdo do *mouse* e arrastando pode-se movimentar o objeto pelos eixos x e y. Através do botão direito do *mouse* tem-se uma janela que permite alterar a escala do objeto (ver figura 33).

**Figura 33 – Botões de escala**

# 8 CONCLUSÕES E EXTENSÕES

## 8.1 CONCLUSÕES

O presente trabalho apresenta os resultados do estudo sobre a criação de um *plug-in* para o *Photoshop*, que permita a transformação de imagens *raster* 2D em *grayscale* para 3D, utilizando a biblioteca gráfica *OpenGL*. Para a representação 3D associou-se a cada um dos pontos da imagem inicial segmentos de retas verticais, cuja altura é associada ao nível de cinza do *pixel* da imagem. Também utilizou para a representação pontos que possibilitam visualizar a imagem em 3D.

Muitas foram as dificuldades para se chegar até o final da construção do protótipo da ferramenta. A falta de conhecimento profundo da linguagem *Visual C++* e o restrito conteúdo encontrado referente aos *plug-ins* do *Photoshop*, apenas uma documentação precária e alguns exemplos de *plug-ins* fornecidos pela *Adobe*.

Um dos grandes motivos que levaram ao desenvolvimento deste protótipo foi o de ser o primeiro *plug-in* brasileiro, segundo pesquisas realizadas em diversas listas de discussões sobre o *Photoshop*, tanto no Brasil como em Portugal, e diretamente no site da *Adobe* ([ADO2000]).

O protótipo realiza de maneira básica e genérica as funções as quais foram propostas, tais como:

a) carregar arquivos *raster* 2D em *grayscale* para dentro do *plug-in*;
b) converter o arquivo *raster* 2D para o formato de arquivo do *OpenGL*;
c) ser capaz de representar em 3D a imagem *raster*;
d) possibilitar a inspeção da imagem 3D em um ambiente de câmera sintética;
e) retornar a imagem 3D do *plug-in* para o *Photoshop*.

Este protótipo também permite dar continuidade aos objetivos iniciais através das propostas de extensões, permitindo ter-se outros produtos que auxiliem à criação de imagens 3D para o *Photoshop*.

## 8.2 LIMITAÇÕES

O *plug-in* não permite uma visualização satisfatória para imagens que não possuem muita alteração de tons de cinza.

Quando é utilizado um *layer* de texto para a conversão 3D, o mesmo não permite a sua visualização, somente uma caixa retangular homogênea; a não ser que o *layer* de texto seja convertido para um *layer* de desenho.

O protótipo funciona corretamente nas seguintes configurações:

a) *Windows 95* ou superior, com suporte ao *OpenGL*;
b) *Adobe Photoshop 5.5*;
c) configuração do vídeo com 32 *bits* de cores;
d) imagem criada no *Photoshop* com 255x260 *pixels* ou superior com a mesma razão para x e y (ex: 510x520 *pixels*) e em *grayscale.*

## 8.3 EXTENSÕES

Como extensão ao protótipo pode-se desenvolver um *plug-in* que suporte outros formatos de imagens, principalmente o RGB, possibilitando assim uma maior manipulação de cores.

Sugere-se aprimorar o algoritmo de geração de imagens 3D, não utilizando somente linhas, mas sim outras primitivas geométricas; para possibilitar uma melhor visualização de imagens que não possuam uma disfusão muito grande de cores.

As funções de obtenção e alteração de informações da imagem realizam uma varredura *pixel* a *pixel* no arquivo das cores válidas. Esta varredura *pixel* a *pixel* necessita de muito processamento do equipamento, esta demora poderia ser minimizada através da obtenção de amostras da imagem através de intervalos fixos ou através de algum algoritmo de pesquisa.

# REFERÊNCIAS BIBLIOGRÁFICAS

[ADO2000] Adobe Systems Incorporated. ***Adobe developers association graphics and publishing SDK***. 2000. Endereço eletrônico: http://partners.adobe.com/asn/developer/gapsdk/main.html

[ALS1995] ALSPACH, Ted. ***Guia incrível do Photoshop***. São Paulo : Makron Books, 1995.

[BRO1996] BROWN, C. Wayne. ***Graphics file formats - reference and guide***. Estados Unidos : Prentice Hal, 1996.

[CUL1997] CULLENS, Chane. ***Usando Visual C++ 4.0***. Rio de Janeiro : Editora Campus, 1997.

[FAL1993] FACON, Jacques. ***Processamento e análise de imagens.*** Província de Córdoba – Argentina : Editora EBAI, 1993.

[FOL1982] FOLEY, Dam A.Van, James D. ***Fundamentais of interactive computer graphics.*** United States : Addision-Weley, 1982.

[KRU1997] KRUGLINSKI, David J. ***Inside Visual C++***. Washington - Estados Unidos : Microsoft Press, 1997.

[LEM1990] LEMOS, Robson Rodrigues. ***Ambiente de visualização 3D***. Universidade Federal do Rio Grande do Sul, Porto Alegre, 1990.

[MCC1999] McClelland, Deke. ***Photoshop 5 & 5.5 – bíblia – edição ouro***. São Paulo : Market Books, 1999.

[PER1989] PERSIANO, Ronaldo Cersar Marinho. ***Introdução à computação gráfica***. Rio de Janeiro : Editora IBPI, 1989.

[PLA1991] PLASTOCK, Roy A. KALLEY, Gordon. ***Computação Gráfica***. Coimbra – Portugal : Mc Graw-Hill, 1991.

[OPE2000] OpenGL.org. ***OpenGL – high performance 2D/3D graphics***. 2000. Endereço eletrônico: http://www.opengl.org.

[REI2000] REIS, Dalton Solano dos. ***Computação gráfica.*** 2000. Endereço eletrônico: http://www.ipa.furb.rct-sc.br/dalton/DiscipCG.

[ROG1985] ROGERS, David F. ***Procedural elementes for computer graphics***. United States : Mc Graw-Hill, 1985.

[SOU2000] SOUZA, Márcio Fulan de. ***Utilizando a biblioteca OpenGL.*** 2000. Endereço eletrônico: http://www.pcs.usp.br/~pcs5748/opengl.

[SWA1994] SWAN, Tom. ***Programação avançada em Borland C++ 4 para Windows***. São Paulo : Berkeley, 1994.

[SIL2000] Silicon Graphics Incorporated. ***The OpenGL website.*** 2000. Endereço eletrônico: http://www.reality.sgi.com/opengl.

[TAI2000] TAINA, Agma. ***Transformações geométricas 2D e 3D.*** 2000. Endereço eletrônico: http://www.gbdi.icmc.sc.usp.br/documentacao/apostilas.

[TRE2000] TREVISAN, Daniela Gorski. ***Computação gráfica.*** 2000. Endereço eletrônico: http://tau.unifran-rs.br/~daniela.

[YAO1992] YAO, Paul, NORTON, Peter. ***Programando em Borland C++ para Windows***. Rio de Janeiro : Berkeley, 1992.