UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SISTEMA PARA VISUALIZAÇÃO 3D USANDO IMAGENS RASTER 2D COM CONCEITOS DE UM AMBIENTE DE VISUALIZAÇÃO CIENTÍFICA

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

FLÁVIO ANDRÉ

BLUMENAU, DEZEMBRO/1999

1999/2-16

PROTÓTIPO DE UM SISTEMA PARA VISUALIZAÇÃO 3D USANDO IMAGENS RASTER 2D COM CONCEITOS DE UM AMBIENTE DE VISUALIZAÇÃO CIENTÍFICA

FLÁVIO ANDRÉ

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. Dalton Solano dos Reis — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Roberto Heinzle

_____

Prof. Antônio Carlos Tavares

DEDICATÓRIA

Dedico este trabalho a meus irmãos Márcio, Daniel, Jonas, Marcos, Davi e Tiago,

as meus amigos Jossué, Adriano, Lúcia

e principalmente a meus pais Argenor e Ivone.

AGRADECIMENTOS

Ao Professor Dalton Solano dos Reis, pela paciência e pelo interesse com o qual orientou este 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.

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.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CG - COMPUTAÇÃO GRÁFICA

RGB - *RED-GREEN-BLUE*

TCL - *TOOL COMMAND LANGUAGE*

VTK - *VISUALIZATION TOOLKIT*

VCi - VISUALIZAÇÃO CIENTÍFICA

RESUMO

Este trabalho visa apresentar um estudo da conversão de imagens *raster* 2D para imagens em 3D, o qual implementará técnicas de câmera sintética em ambiente 3D, incorporando-se ferramenta de Visualização Científica. Apresentará também, considerações sobre aspectos meteorológicos, ambiente de programação Visual C++ e sobre ambiente VTK, objetivando a implementação de um protótipo de software para visualizar imagens em 3D.

ABSTRACT

This work seeks to present a study of conversion of images raster 2D for images in 3D, it will implement techniques of synthetic camera in atmosphere 3D, incorporating tool of Scientific Visualization. It will also present, considerations on meteorological aspects, atmosphere of Visual programming C++ and on ambient VTK, objectifying the implementation of the software prototype to visualize images in 3D.

1 INTRODUÇÃO

Os institutos de meteorologia necessitam de sistemas gráficos que possam fornecer analise de dados interpretados, os quais são representados em 3D, carregando informações de um modo natural e eficiente para o usuário.

Um dos recursos usados pela ciência para prever o tempo (condições meteorológicas) é o uso de satélites, estimativas de precipitação, obtenção do campo de velocidade do vento através do deslocamento das nuvens, temperatura da superfície da Terra (por exemplo para o acompanhamento de geadas) e da superfície do mar (para a localização de cardume de peixes, perfis verticais de temperatura e umidade, entre outros) [GRA91].

O Instituto de Pesquisas Ambientais (IPA) da Universidade Regional de Blumenau (FURB) recebe, via Internet, imagens dos satélites Meteosat e Goes (Figura 01). Estas imagens são analisadas e comparadas a fim de obter-se um acompanhamento meteorológico. Os recursos computacionais atualmente disponíveis, em nível de software, para permitir tal acompanhamento meteorológico são feitos por editores gráficos (PaintBrush, Paint Shopp Pro e outros). Estes por sua vez, como se tratam de aplicativos para necessidades genéricas em editoração gráfica, não permitem ter-se um ambiente apropriado a um acompanhamento meteorológico.

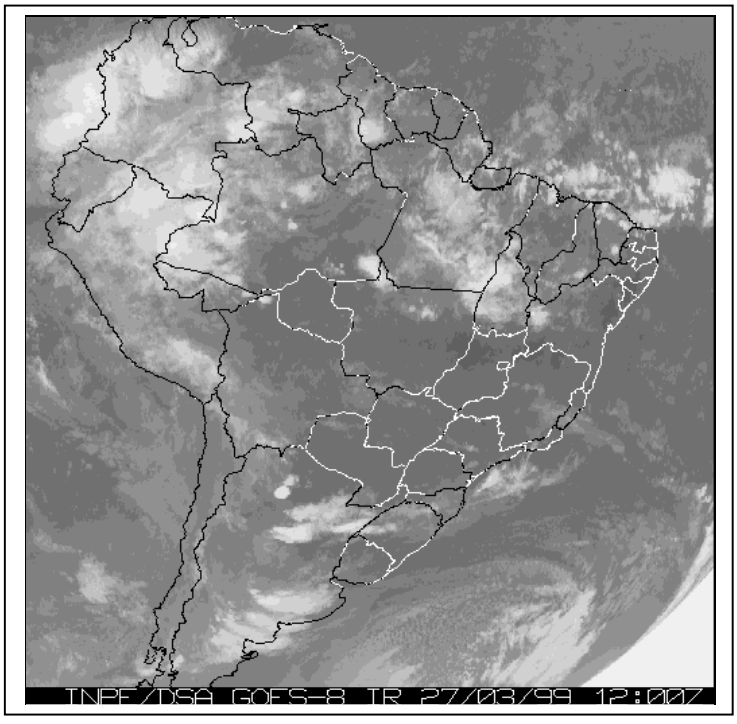
O entendimento dos sistemas meteorológicos atuantes em uma determinada região é um pré-requisito essencial para a geração de prognósticos com maior segurança, precisão e objetividade. O estudo sistemático de cada estágio do ciclo de vida médio destes sistemas é uma das formas de melhor compreendê-los e avaliá-los.

A necessidade de realizar acompanhamentos cada vez mais precisos e otimizados está diretamente associada ao tratamento, manipulação e visualização de dados. Estas ações devem ser capazes de auxiliar a caracterização dos sistemas, levando-se em

consideração os seus mais diversos parâmetros. Evidencia-se, portanto, a relação entre este tipo de estudo e a área de Visualização Científica.

Desta forma, o presente trabalho de conclusão de curso apresentará um estudo sobre técnicas de câmera sintética em ambiente 3D usando imagens de satélite raster 2D para imagens em 3D aplicada à Meteorologia, incorporando ferramenta de visualização científica com a utilização do VTK em um ambiente de programação orientado a objeto (*Visual C++*).

Figura 1 - Imagem do satélite Goes



Fonte: [SMI99]

1.1 MOTIVAÇÃO

A maior motivação que me levou a desenvolver este trabalho é o resultado que irá trazer para minha vida profissional, com este protótipo concluído, irei obter um retorno para minha empresa que tem como um objetivo maior trabalhar com este software para que ele se torne um software vendável para usuários que necessitem de representação de imagens em 3D.

1.2 OBJETIVO

O objetivo deste TCC será o desenvolvimento de um protótipo de uma ferramenta de Visualização Científica, sendo esta capaz de carregar arquivos gráficos (imagens de satélite) raster, incorporando-se informações 3D.

1.3 ORGANIZAÇÃO DO TEXTO

O primeiro capítulo fornece uma introdução ao trabalho desenvolvido, mostrando o modelo atual e apresentando o que pode ser alcançado através deste trabalho.

O segundo capítulo traz uma base teórica sobre o meteorologia, como funcionam os satélites, como são obtidas as imagens de satélite e suas principais utilidades.

No terceiro capítulo fornece uma visão das técnicas de Computação Gráfica e formatos de imagens utilizadas no desenvolvimento do protótipo.

No quarto capítulo fornece uma introdução ao Visual C++, trazendo algumas das características básicas do ambiente de programação.

No quinto capítulo fornece uma introdução ao Visualization ToolKit, mostrando alguns exemplos de utilização da classes do VTK.

No sexto capítulo é apresentado as especificações do protótipo, englobando o seu funcionamento e aspectos de implementação.

O sétimo capítulo faz uma análise conclusiva sobre o trabalho, dificuldades encontradas e como solucioná-las, e futuras extensões do trabalho que poderão ser realizadas.

2 ASPECTOS METEOROLÓGICOS

2.1 FUNDAMENTAÇÃO

A meteorologia é uma ciência naturalmente quadri-dimensional, já que um de seus objetivos básicos é a quantificação da estrutura dinâmica da atmosfera terrestre, visando prognosticar determinados comportamentos com certa precisão. A inserção do fator tempo como quarta dimensão permite obter-se uma visualização dinâmica da atmosfera. Com este tipo de abordagem a representação meteorológica torna-se um forte instrumento para pesquisas na área de Visualização Cientifica.

O prognóstico do tempo começa nas estações meteorológicas, onde são registradas as condições atmosféricas, a partir das quais são feitas as previsões. Os centros meteorológicos de todo o mundo trocam informações, assim, todos os países podem ter uma imagem meteorológica do globo. O intercâmbio é promovido pela organização meteorológica Mundial (OMN), criada em 1947 sob o patrocínio da ONU. A organização tem seções regionais sediadas em Nairobi (Quênia), Rio de Janeiro (Brasil), Arlington (Estados Unidos), Rugby (Inglaterra) e Guam (ilha do Pacífico) [CIV76].

As informações mais importantes das estações meteorológicas referem-se à temperatura, à pressão, à umidade relativa do ar, à quantidade de precipitações pluviométricas, à força e direção dos ventos. Mas também registram dados acerca da quantidade, altura e tipo de nuvens, visibilidade, etc. [CIV76].

Os instrumentos utilizados para obtenção das informações são:

- termômetro de máximas e mínimas para medir temperatura;
- termógrafo também para medir temperatura;
- barômetros metálicos para registrar a pressão;
- barígrafo também para registrar a pressão;
- psicrômetro para obtenção da umidade relativa do ar;
- pluviômetro para medir o nível de precipitações;
- pluviógrafo também para medir o nível de precipitações;
- anemômetro para cálculo da força e direção do vento;

- balões meteorológicos para pesquisa pormenorizado das condições meteorológicas em níveis superiores da atmosfera através de aparelhos automáticos;
- satélites equipados para obtenção de diversas informações de grandes áreas da superfície do globo terrestre.

As imagens são geradas através das informações obtidas pelos aparelhos do satélite e possibilitam acompanhar a situação sinóptica na área de cobertura do mesmo, assim como fazer estimativas de precipitação, obtenção do campo de velocidade do vento através do deslocamento das nuvens, temperatura da superfície da terra e da superfície do mar. Para maiores detalhes de obtenção das imagens ver capítulo 4 em [DOR97].

Estas imagens são analisadas através do conhecimento técnico do responsável pela análise e através de tabelas que possuem informações sobre vários aspectos relacionados ao clima e às condições do tempo. Essas tabelas possuem a relação de cada cor com os valores das diversas variáveis utilizadas na análise meteorológica, tais como temperatura, pressão do ar, umidade relativa do ar, etc [DOR97].

2.2 IMAGENS DE SATÉLITE

De acordo com [CRO93] as imagens de sensoriamento remoto são constituídas por um arranjo de elementos sob a forma de uma malha ou *grid* (matriz). Cada célula desse *grid* tem sua localização definida de acordo com um sistema de coordenadas do tipo coluna e linha, representados por x e y, respectivamente. O nome dado a essas células é *pixel*, derivado do inglês *picture element*. Para um mesmo sensor remoto, cada *pixel* corresponde sempre a uma área com as mesmas dimensões na superfície da Terra. Cada *pixel* possui também um atributo numérico z, que indica o nível de cinza representando a intensidade de energia eletromagnética medida pelo sensor, para a área da superfície terrestre correspondente.

Uma imagem de satélite pode ser vista então como uma matriz, de dimensões x colunas por y linhas, com cada elemento possuindo um atributo z (nível de cinza). No caso

das imagens de sensoriamento remoto, essas matrizes possuem dimensões de até alguns milhares de linhas e de colunas [CRO93].

Segundo Merege [MER97] os sistemas orbitais de sensoriamento remoto em operação podem ser classificados de acordo com sua principal aplicação em:

- satélites meteorológicos,
- satélites de recursos naturais, e
- satélites de aplicação híbrida.

A principal diferença entre os satélites tipicamente meteorológicos e os demais é o tipo de órbita, que neste caso é geosíncrona[*]. Com isto o sensor coleta dados constantemente de uma mesma área da superfície terrestre. No caso deste trabalho foram utilizados os satélites Goes e Meteosat sendo estes posicionadas de maneira que coletam informações sobre o Brasil e regiões próximas.

O sistema de Satélites Meteorológicos responsável pelo monitoramento do tempo é composto pelos satélites geoestácionários e polares nas tabelas 01 e 02 [GRA91] apresentados abaixo.

Um satélite é geoestacionário quando seu período de rotação em torno da Terra coincide com o período e sentido de rotação da própria Terra, ou seja, aproximadamente uma órbita a cada 24 horas, de oeste para leste. Desta forma, com relação a um observador na Terra, a posição de um satélite geoestacionário permanece inalterada com o decorrer do tempo.

[*] geosíncrona - posicionada no Equador com velocidade angular igual à velocidade de rotação da Terra.

Tabela 01 – Satélites Meteorológicos Geoestacionários

Nome	Localização	Cobertura	Gerenciado
Goes – W	135^{o} W	Pacífico Leste América do Norte	EUA
Goes – E	75^{o} W	América do Norte América do Sul Oceano Atlântico	EUA
Meteosat – 4	0^{o}	Europa África Oceano Atlântico	EUROPA
Meteosat – 3	50^{o} W	Oceano Índico Costa do Brasil	EUROPA
INTSAT	74^{o} W	Ásia (Mid. Ásia) Oceano Índico	ÍNDIA
GMS – 3	140^{o} E	Pacífico Oeste Sudeste da Ásia Austrália	JAPÃO
Altitude de 35800 Km sobre o equador e estabilizados por rotação – 100 rpm			

Tabela 02 – Satélites Meteorológicos de Órbita Polar (em operação)

Satélites	Órbita heliossíncrona
Série NOAA NOAA 10 e NOAA 11	Aproximadamente polar, altitude de 850 Km e período de 101 min.
METEOR 2	Aproximadamente polar, altitude de 900 Km e período de 102 min.

Os satélites polares fazem a cobertura de toda a superfície terrestre, registrando o mesmo ponto da superfície duas vezes ao dia. Como foi visto, os satélites geoestácionários não fazem a cobertura das regiões polares e fornecem imagens do disco todo, visto de sua posição a cada 30 minutos.

2.3 SATÉLITE GOES

O sistema Goes consiste de dois satélites operacionais, o Goes E e Goes W localizados a 75^{o} e 135^{o} de longitude oeste respectivamente. Estes satélites permitem fazer um acompanhamento durante 24 horas de fenômenos meteorológicos que ocorrem na sua área de cobertura [DOR97].

2.3.1 FORMA DO SATÉLITE

Na configuração externa do Goes existem antenas para comunicação (entre satélite/terra/satélite), sensor solar, magnetômetro e abertura para o sensor de observação do tempo, o VAS (sondador atmosférico), o VISSR – *Visible Infrared Spin-Scan Radiometer* (Radiômetro Visível e Infravermelho e Varredura por rotação). O sondador VAS é a forma aperfeiçoada do VISSR e tornou-se operacional a partir do Goes 4, lançado em 09/09/1980 [GRA91].

Atualmente o Goes tem capacidade de receber e retransmitir dados de plataformas de coletas de dados, transmitir cartas sinóticas (diagnósticos e prognósticos) e imagens de baixa resolução (WEFAX), monitorar o campo magnético, medir o fluxo de partículas energéticas, determinar o fluxo de raio X proveniente do Sol na vizinhança do satélite e conta também com o sistema internacional de busca e salvamento por satélites (SARSAT) [GRA91].

2.3.2 AQUISIÇÃO DE IMAGENS

Segundo [GRA91] “A aquisição das imagens ou sondagens é feita através do VAS (*VISSR Atmospheric Sounder*), onde estão localizados o conjunto de espelhos e sensores para medidas da radiação nos espectros infravermelho e visível. A cada giro do

satélite o sensor varre a terra no sentido oeste-leste, sendo que a varredura, no sentido norte-sul, é feita incrementando-se o espelho de varredura de um ângulo fixo (192 uradianos)".

Os detetores são alojados no plano focal conforme um padrão pré-determinado, de tal modo que cada detentor defina um campo de visada. A radiação entra no telescópio num ângulo de 90^{o} em relação ao seu eixo ótico. O mecanismo de deslocamento angular por passos, facilita a varredura terrestre. Tendo em vista que o sentido de rotação do satélite em torno de seu eixo é contrário ao da Terra, após a varredura dos sensores este é incrementado de 109 u, permitindo então, a formação da imagem no sentido norte-sul. A amostragem no sentido oeste-leste é de 0,21 rad. para o visível e de 0,84 rad. para o infravermelho [GRA91].

3 FUNDAMENTOS DE COMPUTAÇÃO GRÁFICA

A computação gráfica 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 [PER89]:

- Síntese de imagens: que se ocupa da produção de representações visuais a partir de especificações geométrica e visual de seus componentes;
- 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;
- 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 visualizadas captadas por recursos os mais diversos: digitalização de fotos, tomadas de uma camêra de vídeo, ou imagens de satélite. 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, digamos, por um equipamento de vídeo, procura identificar os elementos que a compõem e determinar suas especificações.

3.1 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 [FAL93].

3.1.1 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 [BRO96].

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

- *Raster Device* - Qualquer dispositivo que produz um quadro exibindo uma ordem de amostras. Alguns exemplos incluem, televisão, impressoras de ponto-matriz, impressoras de tinta-jato, e impressoras de laser.
- *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*.
- *Voxel* - Uma única amostra de um raster 3D de dados. *Voxel* é um abreviação para elemento de volume.
- *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.
- *Pixel Map* - Memória usada para armazenar imagens raster 2D.

O formato GIF é do tipo raster, também muito utilizado pelos centros meteorológicos.

3.1.2 GIF

O acrônimo de GIF é (*Graphics Interchange Format*) Formato de Gráficos interativos, originário da CompuServer Incorporated, criado por Steve Whilhire onde teve como motivação do projeto as informações de códigos baseados em telefones. Transferindo de maneira eficientes as imagens em linhas telefônicas onde a necessidade de critérios mais apurado de formato de arquivo. As principais versões no mercado são as versões *Version87a* de maio de 1987 e a *Version89a* de julho de 1989.

Amarzena dados do tipo r*aster 2D*, armazenados em binários, os dados são organizados seqüencialmente, as representação de cores são definidos numa tabela de 256 cores (*Gray Scale*).

3.2 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) [TAI99].

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

Figura 2 - Sistema de Coordenadas dado pela Regra da Mão direita

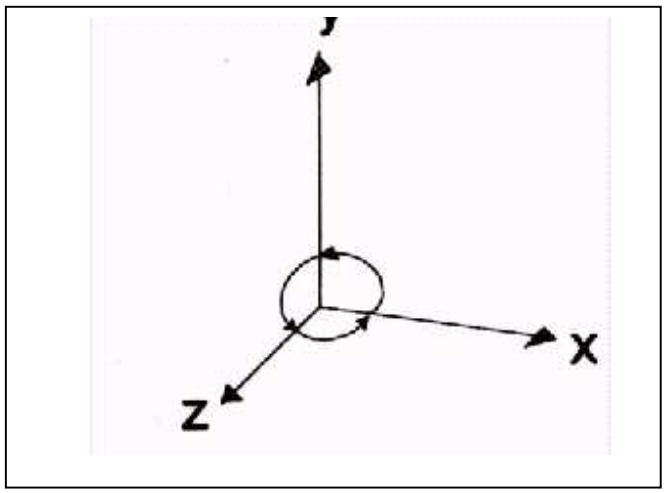


Tabela 3 - Rotação

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, ou seja (ver quadro 1):

Quadro 1 - Equação (1a)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Assim, a equação (1a) pode ser representada também como (quadro 2):

Quadro 2 – Fórmula (1a)

P' = T(dx, dy, dz)*P

Similarmente a ESCALA em 3D, fica (ver quadro 3 e quadro 4):

Quadro 3 – Equação (2a)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Quadro 4 - Fórmula (2a)

P' = S(sx,sy,sz) * P

Finalmente, verifiquemos 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 (ver quadro 5):

Quadro 5 – Equação (3a)

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

A matriz de rotação em relação ao eixo x é (ver quadro 6):

Quadro 6 – Equação (4a)

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

A matriz de rotação em relação ao eixo y é (ver quadro 7):

Quadro 7 – Equação (5a)

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

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 uma matriz da seguinte forma (ver quadro 8).

Quadro 8 – Equação (6a)

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

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 (ver quadro 9).

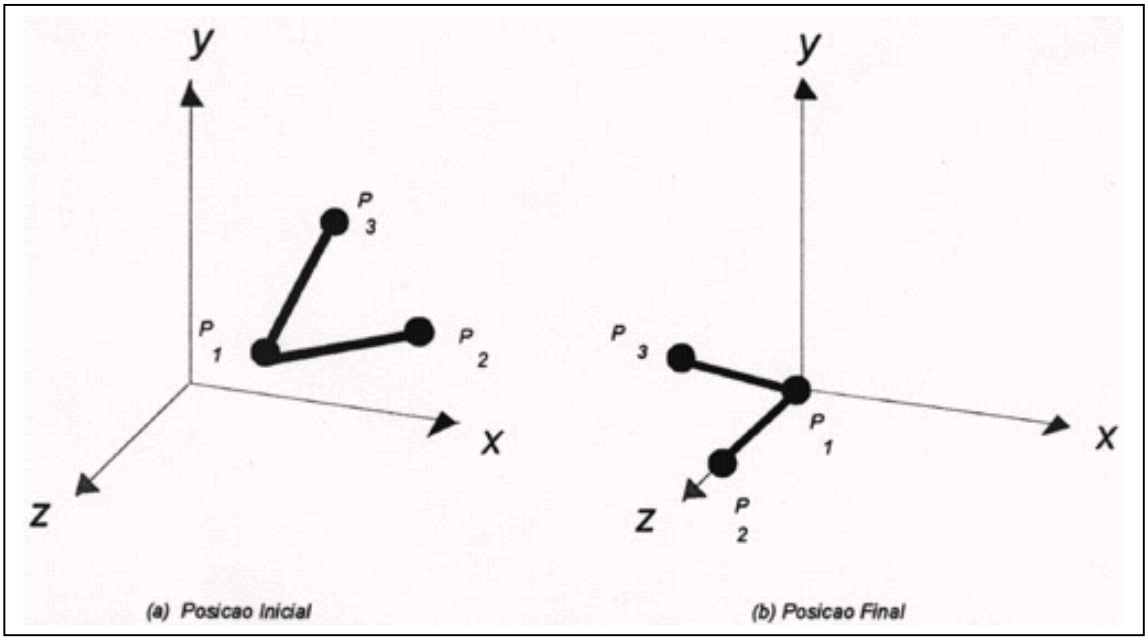
Quadro 9 – Equação (7a)

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

3.3 COMPOSIÇÃO DE TRANSFORMAÇÕES EM 3D

A composição de transformações em 3D pode ser entendida mais facilmente através do exemplo indicado na figura 3. 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 [TAI99].

Figura 3 – Transformação 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, teremos os seguintes quatro passos:

1. Transladar P_1 para a origem.
2. Rotacionar o segmento P_1P_2 em relação ao eixo **y**, de forma que ele (P_1P_2) fique no plano **y-z**.
3. Rotacionar o segmento P_1P_2 em relação ao eixo **x**, de forma que ele (P_1P_2) fique sobre o eixo **z**.
4. Rotacionar o segmento P_1P_3 em relação ao eixo **z**, de forma que ele (P_1P_3) fique no plano **y-z**.

Primeiro Passo: Transladar P_1 para a Origem.

A equação de translação é (ver quadro 10):

Quadro 10 – Equação (8a)

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

Aplicando T a P_1, P_2 e P_3, temos:

- Inicialmente,

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

- Mais,

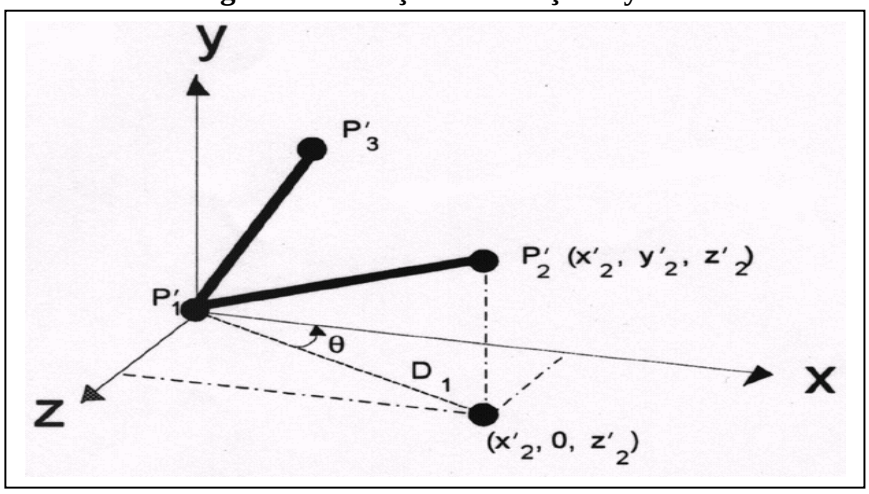
$$P_2 = T(-x_1,-y_1,-z_1).P_2 = \begin{bmatrix} x_2-x_1 \\ y_2-y_1 \\ z_2-z_1 \\ 1 \end{bmatrix}$$

- finalmente

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

Segundo Passo: Rotacionar em Relação ao eixo Y (Figura 4).

Figura 4 - Rotação em relação a y



A figura 4 (o ângulo θ indica a direção positiva de rotação em relação a **y**. O ângulo utilizado na realidade é -(90-θ)) mostra P_1P_2 após o primeiro passo, bem como a projeção de P_1P_2 no plano **x-z**. O ângulo de rotação é -(90-θ)=θ-90. Então:

$$\sin(\theta - 90) = -\cos\theta = -\frac{x_2}{D_1} = \frac{x_2 - x_1}{D_1},$$

$$\cos(\theta - 90) = \sin\theta = \frac{z_2}{D_1} = \frac{z_2 - z_1}{D_1},$$

onde

$$D_1 = \sqrt{(z_2)^2 + (x_2)^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}$$

Então substituindo estes valores na equação 5a (ver quadro 7), temos:

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

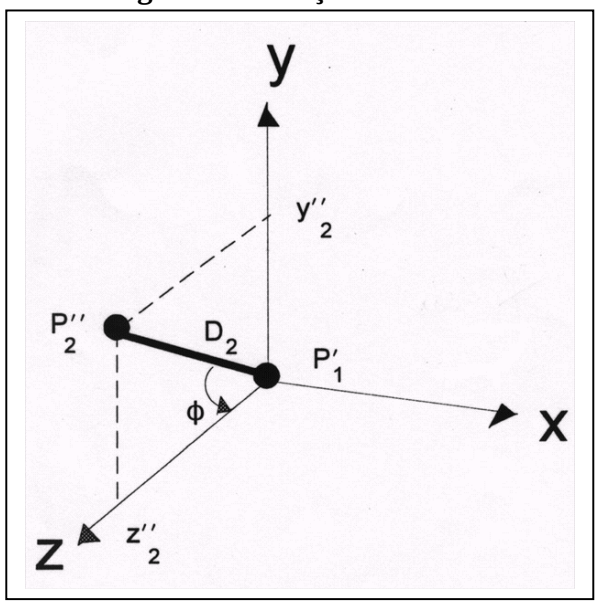
Como era esperado a componente *x* de P_2'' é zero, e *z* possui comprimento D_1.

Terceiro Passo: Rotacionar em Relação ao eixo X

A figura 5 mostra P_1P_2 após o segundo passo (o segmento de reta 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

$$\cos\phi = \frac{z_2}{D_2}, \quad \sin\phi = \frac{y_2}{D_2}$$

Figura 5 - Rotação no eixo X



onde $D_2 = |P_1'' P_2''|$ é o comprimento do segmento de reta $P_1''P_2''$. Como as translações e rotações preservam o comprimento, o comprimento de $P_1''P_2''$ é o mesmo de P_1P_2, ou $D_2 = |P_1'' P_2''| = |P_1 P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

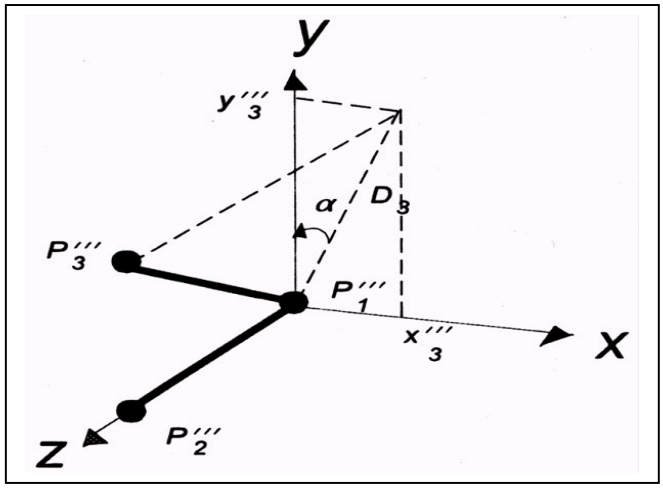
O resultado da rotação no terceiro passo é:

$$P_2''' = R_x(\phi).P_2'' = R_x(\phi).R_y(\theta - 90).P_2'$$
$$= R_x(\theta).R_y(\theta - 90).T.P_2 = [0 \quad 0 \quad |P_1P_2| \quad 1]^T$$

Dessa forma, agora P_1P_2 agora está sobre (coincidindo) o eixo **z** positivo.

Quarto Passo: Rotacionar em Relação ao eixo Z (figura 6).

Figura 6 - Rotação no eixo Z



A figura 6 mostra P_1P_2 e P_1P_3 após o terceiro passo, com P_2''' sobre o eixo **z** e

$$P_{3'} = [\, x_{3'} \;\; y_{3'} \;\; z_{3'} \;\; 1 \,] = R_x(\phi).R_y(\theta - 90).T(-x_1, -y_1, -z_1).P_3 \quad .$$

P_3''' em

A rotação é dada pelo ângulo positivo α, e

$$\cos\alpha = \frac{y_{3'}}{D_3}, \quad \sin\alpha = \frac{x_{3'}}{D_3}, \quad D_3 = \sqrt{x_{3'}^{\,2} + y_{3'}^{\,2}} \quad .$$

$P_1'P_3'$ é trazido para o plano y-z.

Dessa forma, após o quarto passo o resultado é alcançado como visto na figura 3 (b).

A matriz de composição M é a transformação necessária à composição

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

solicitada na figura 3.

Como podemos ver a base para transformação e a composição de transformações 3D estão fundamentalmente ligadas a operações em matrizes, permitindo a definição de uma camêra sintética imprescindível à um ambiente de Visualização Científica.

3.4. VISUALIZAÇÃO CIENTÍFICA

Visualização é uma parte de fora da vida cotidiana. Mapas de tempo, gráficos em 3D, indústria de entretenimento, todas estas áreas utilizam a visualização em abundância. Informalmente, visualização é a transformação dos dados ou informações em quadros. Visualização empenha o aparato sensório humano primário, visão, como também o poder de processo da mente humana e imagens meteorológicas. Os resultados são muito simples e efetivo como meio de comunicar informação complexa [SCH96].

Diferentes terminologias são usadas para descrever visualização científica tais como o nome formal dado a área em computação que trabalha com interface de usuário, representação de dados e algoritmos processando representações visuais, e outra apresentação sensória como som ou toque [SCH96].

Visualização de dados é outra frase que descreve visualização. Embora não foram aceitadas definições rigorosas, visualização de dados geralmente conota as aplicações de métodos estatísticos fora do reino da visualização. Por outro lado, visualização de dados implica aplicações além das ciências. A visualização científica é muito estreita como técnicas de visualização, mas movimentam além do domínio científico as áreas de negócios, ciência social, demográficas e administração de informação em geral [SCH96].

4 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. Que reúne um conjunto de ferramentas como [CUL97]:

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

4.1 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 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 [CUL97] [KRU97].

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 4.

Tabela 04 - 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 .MDP.

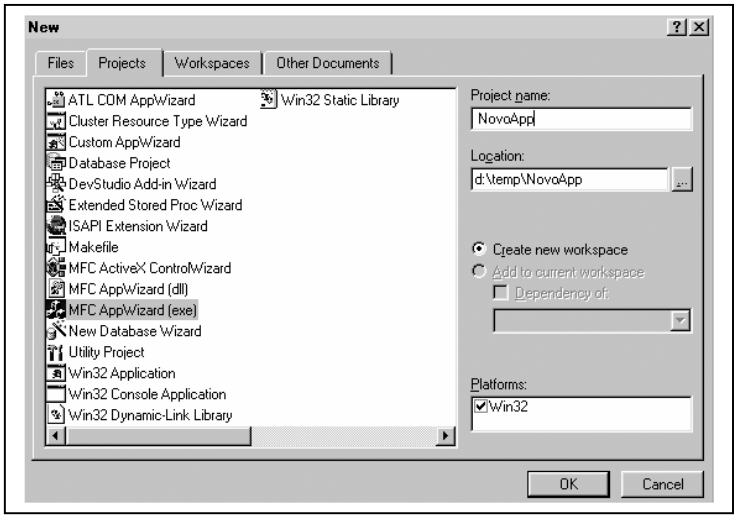
4.1.1 A CRIAÇÃO DE APLICATIVOS COM O APPWIZARD

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 e incorpora suporte para recursos, como Windows Sockets ou OLE 2.0.

Quando se utilizada, o AppWizard gera-se um projeto rico em recursos, que contém todos os arquivos necessários a um aplicativo básico. Para criar-se um novo projeto e acessar o AppWizard, escolha-se File,New.

A caixa de diálogo New aparece, escolha-se Project Workspace na lista de opções disponíveis. A caixa de diálogo New Project Workspace aparece (figura 7) na caixa de texto Name, digita-se **NovoApp** e então dá-se um clique no botão Create.

Figura 7 - A caixa de diálogo New Project Workspace



Neste ponto, ira-se passar por uma série de seis passos, cada qual fazendo-se perguntas sobre o aplicativo que deverá ser criado (ver Figura 8). Para aceitar o valor padrão, dá-se um clique no botão Next através dos seis passos. Em seguida, dá-se 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, 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 9). Quando a compilação estiver terminada, escolha-se Build, Execute (ou pressiona-se Ctrl+F5).

Figura 8 - Passo do MFC AppWizard

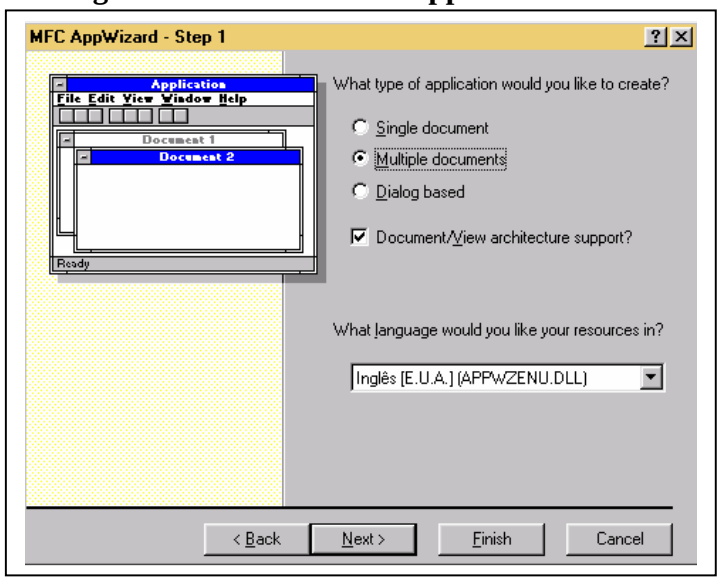
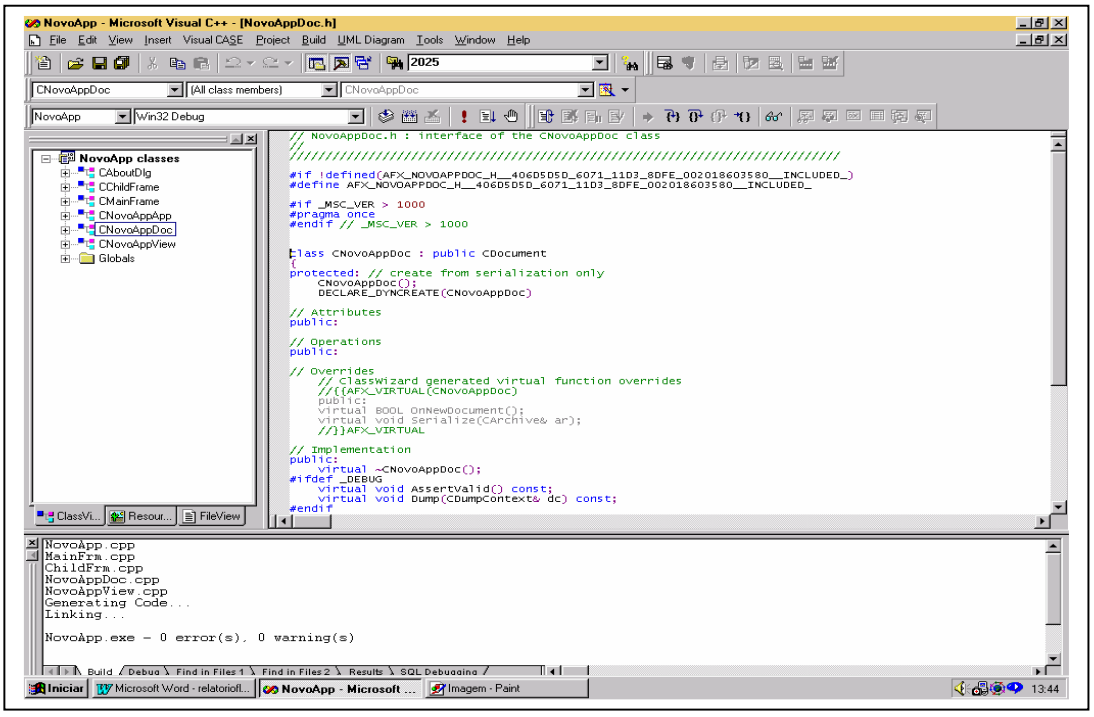


Figura 9 - A janela de saída da compilação



4.1.2 COMO A MFC É FORNECIDA

A MFC é fornecida sob a forma de classes de C++. Essas classes contêm todos os detalhes de como lidar com a API do Windows, incluindo alguns assuntos como a OLE (*Object Linking and Embedding*), a ODBC (*Open Database Connectivity*) e o Winsok (a implementação do Windows do TCP/IP). As classes fornecem a base para a escrita de um aplicativo para Window [CUL97].

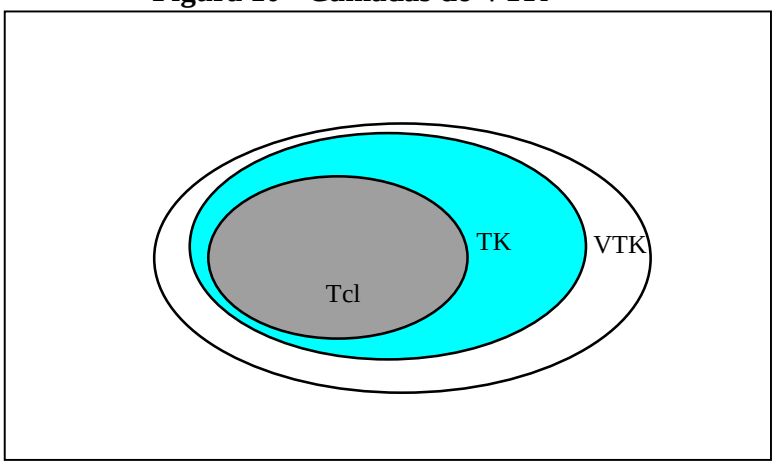
A estrutura do MFC fornece suporte para aplicativos que usam janelas isoladas (também conhecidas como aplicativos SDI, ou de interface de documento único) ou múltiplas janelas dentro de uma janela de aplicativo (conhecidos como aplicativos MDI ou de interface de vários documentos).

A MFC encapsula todos os elementos da interface gráfica de dispositivos (GDI) do Windows e da interface com o usuário. A MFC permite a criar objetos caneta, fontes, pincéis, entre outros, com as classes relevantes destruindo o objeto da GDI quando ele sai do escopo. De modo similar, a MFC fornece classes para botões (incluindo aqueles com *bitmaps* personalizados), caixas de listagem, caixas de combinação, botões de rádio e campos de entrada de texto. A estrutura também fornece suporte total para o uso de caixas de diálogos comuns, que são o padrão fornecido pela Microsoft para a abertura de arquivos.

5 VISUALIZATION TOOLKIT

VTK é um *toolkit* para Visualização, ou seja, é uma biblioteca de rotinas que executa funções gráficas e de visualização. VTK é projetado com base na metodologia orientada a objetos e escrito em C++. VTK pode ser programado usando C++ ou Tcl/Tk. Uma camada de código cria uma interface entre as funções de VTK e Tcl/TK. Dessa forma, todas as funções do VTK estão disponíveis juntamente com as to Tcl e TK, conforme ilustrado na Figura 10 [SCH96].

Figura 10 - Camadas do VTK



Assim, a partir do ambiente VTK, tem-se acesso a todas as funções do VTK.

5.1 ORIENTAÇÃO A OBJETOS

O VTK é desenvolvido segundo a abordagem orientada a objetos. Utilizando essa abordagem, um sistema modela *classes* de entidades, para todas as entidades presentes no sistema. Tais entidades são imbuídas de significado, atributos e métodos, de acordo com seu comportamento no mundo real. Seu *significado* representa o papel que desempenha no funcionamento do sistema. Seus *atributos* são propriedades ou parâmetros das entidades, e

os métodos alteram seus atributos e determinam suas ações. A programação orientada a objetos difere da programação convencional, em que a execução de uma tarefa implica na execução de um método de uma entidade relacionada com aquela tarefa [SCH96].

O exemplo a seguir mostra as diferenças de programação entre as abordagens convencional e orientada a objetos:

Em C:

```
Primitive *aPrim;
DrawPrimitive (aPrim);
procedure DrawPrimitive (aPrim)
{
      if (aPrim->type == TRIANGLE) then DrawTriangle (aPrim)

            else if (aPrim->type == SQUARE) then DrawSquare (aPrim)

          else if (aPrim->type == CIRCLE) then DrawCircle (aPrim)

          }
```

Em C ++:

```
aPrim->Draw();
```

A criação de objetos é feita por instanciação das classes definidas, de maneira que cada novo objeto tem seus próprios métodos, que podem ser encarados como cópias dos métodos da classe original.

Assim, por exemplo, pode-se definir um objeto do VTK pertencente a classe `vtkActor`, utilizando o seguinte comando:

```
VtkActor aBall;
```

5.3 CONCEITOS BÁSICOS DE VTK

Aqui são apresentados alguns conceitos básicos que vão orientar a familiarização inicial com o VTK.

5.3.1 CENAS, ATORES E RENDERING

Para se apresentar objetos (*rendering*) em computação gráfica é necessário definir uma cena. Em VTK a cena é definida, criando uma '*RenderWindow*' e um '*Renderer*'. Objetos, chamados atores (*Actors*) são definidos e adicionados à cena, que é mapeada e apresentada quando o programador desejar [SCH96].

Para fazer os gráficos, o VTK usa uma biblioteca do tipo OpenGL, e possui objetos para que o programador possa ter acesso a algumas dessas funções gráficas diretamente.

De uma maneira geral, os passos para 'visualizar' uma cena são:

1. Criar os dados a serem visualizados,
2. Filtrar os dados para as técnicas desejadas,
3. Criar uma janela de 'rendering',
4. Criar um renderer,
5. Mapear os dados para o sistema de 'rendering' (biblioteca gráfica),
6. 'Renderizar' a cena,
7. Opcionalmente, criar um 'interactor' (objeto responsável por manipulação interativa do resultado).

A seqüência de passos de 3 a 7 acima é denominada '*pipeline*' de renderização (*rendering pipeline*) por ser responsável pela geração da imagem na tela.

Os objetos básicos do VTK encarregados de realizar o *rendering* são:

- `vtkRenderWindow` - gerencia uma janela no periférico de apresentação; um ou mais 'renderizadores' desenham em uma instância de `vtkRenderWindow`.
- `vtkRenderer` - é o renderizador. Coordena o processo de *rendering* envolvendo luzes, câmeras e atores.

- `vtkLight` - uma fonte de luz para iluminar a cena.
- `vtkCamera` - define uma posição de observação, ponto focal, e outras propriedades visuais da cena.
- `vtkActor` - representa um objeto ‘renderizado’ numa cena, incluindo suas propriedades e posição no sistema de coordenada do mundo real.
- `vtkProperty` - define as propriedades de aparência de um ator incluindo cor, transparência, e propriedades de iluminação como coeficientes de reflexão especular e difusa. Também define propriedades de representação, como *wireframe* ou superfície sólida.
- `vtkMapper` - a representação geométrica de um ator (*Actor*). Mais do que um ator pode referenciar o mesmo *mapper*.

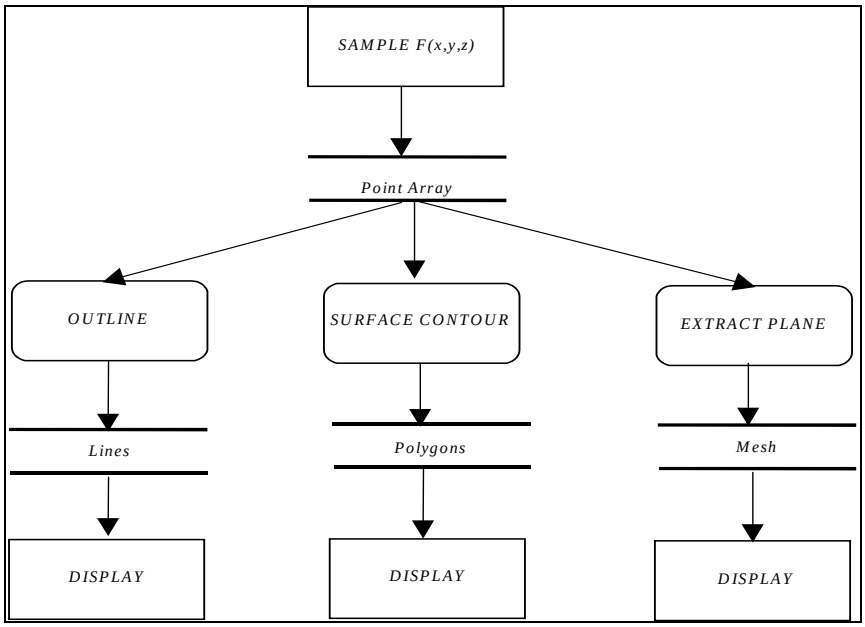
5.4 ‘PIPELINE’ (OU ‘NETWORK’) DE VISUALIZAÇÃO

O *Pipeline* de Visualização do VTK compreende os objetos que podem ser conectados de maneira a gerar a visualização desejada [SCH96] (ver quadro 11).

Os elementos de um *Pipeline* incluem três tipos de Objetos:

- *sources* (fontes) - Geram dados. Fazem leitura de dados e geram pontos a partir de funções implícitas.
- *filters* (filtros) - Transformam dados. Extraem informações e dados a partir de outros dados.
- *mappers* (mapeadores) - Representam a parte final do fluxo de visualização. Usados para converter dados em primitivas gráficas ou escrever gráficos em arquivos.

Quadro 11 - Exemplo de *Pipeline*

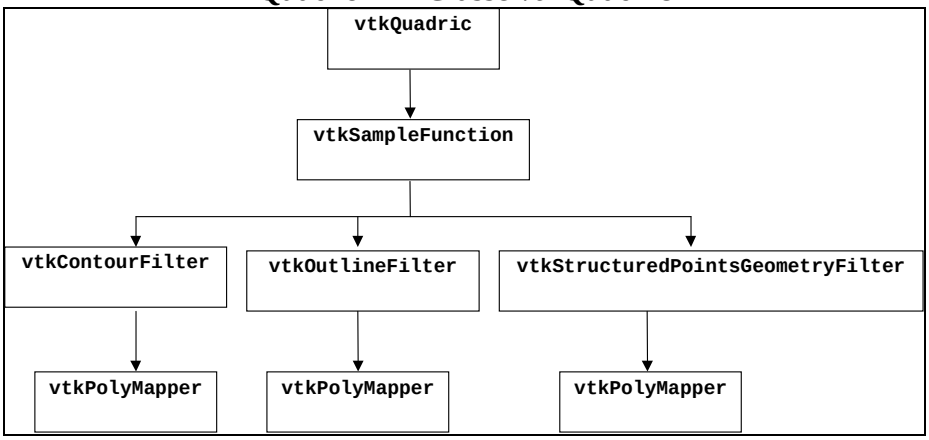


5.4 CONEXÕES ENTRE OBJETOS NO VTK - TIPOS DE DADOS

Um aspecto importante o *Pipeline* de Visualização é saber quais objetos podem ser conectados entre si. A melhor maneira de se fazer isso é, após escolher as tarefas de visualização a serem executadas, verificar que tipos de dados são produzidos como saída e que tipos de dados são aceitos como entrada dos filtros, fontes e mapeadores [SCH96].

Um *pipeline* (ou *network*) simplificado para o exemplo acima seria (ver quadro 12).

Quadro 12 - Classe vtkQuadric



Nesse caso, o objeto `vtkSampleFunction` é um objeto destinado a amostrar funções implícitas, e retorna a amostragem num grid 3D, do tipo `vtkStructuredPoints`.

Por sua vez, os objetos `vtkContourFilter`, `vtkOutlineFilter` e `vtkStructuredPointsGeometryFilter` aceitam como entrada objetos do tipo `vtkStructuredPoints`, e isso implica que suas entradas podem ser conectadas à saída do objeto `vtkSampleFunction`. Esses três filtros produzem como saída poligonais (do tipo `vtkPolyData`), que é o tipo de entrada que o objeto `vtkPolyMapper` aceita para produzir as primitivas gráficas necessárias. Assim, uma vez que se deseja utilizar um certo tipo de filtro, é necessário compatibilizar as entradas que eles necessitam, e as saídas que eles produzem com os demais filtros e as demais fontes do *pipeline*.

Em circunstâncias onde se deseja utilizar um filtro a partir de um certo tipo de dados que ele não aceita, existem duas possibilidades:

- caso1 - encontrar um filtro que faça exatamente a mesma coisa e que trabalhe com o tipo de dados desejado; ou
- caso 2 - encontrar um filtro adicional que gere o tipo de dados desejado a partir do tipo de dados disponível (um conversor).

Um exemplo do caso 1 é o filtro `vtkStructuredGridGeometryFilter`, que executa o mesmo tipo de operação executada pelo filtro do exemplo, porém admitindo como entrada o tipo de dados `vtkStructuredGrid`. Assim, se houver necessidade da mesma operação para um outro tipo de dado, pode-se encontrar um filtro compatível.

No caso 2, existem vários filtros que extraem tipos ou geometria de outros tipos. Exemplos são:

- `vtkDataSetToDataSetFilter`,
- `vtkDataSetToPolyFilter`,
- `vtkDataSetToStructuredGridFilter`,
- `vtkImageToPolyDataFilter`,
- `vtkDataSetToStructuredPointsFilter`,
- `vtkDataSetToUnstructuredGridFilter`,
- `vtkStructuredGridToPolyFilter`.

5.4.1 TIPOS DE OBJETOS NO VTK

A discussão acima também levanta outro aspecto do VTK: os tipos de objetos que ele implementa. Existem basicamente dois tipos de objetos [SCH96]:

- Objetos de Dados (*Data Objects*): São objetos que implementam os tipos de dados: *grids* (malhas), valores, vetores, tensores, metadados (como luzes, escalas, transformações) e muitos outros.
- Objetos de Processo (*Process Objetcs*): São Objetos que representam processos de criação e transformação de dados.

5.4.2 CONECÇÕES: TIPO E MULTIPLICIDADE

Duas características controlam as conexões entre objetos no VTK: os tipos de dados fluindo pelo sistema e a multiplicidade de entrada e saída dos objetos [SCH96].

Objetos só podem ser conectados se tiverem tipos compatíveis. Por exemplo, no comando.

Em C++:

```
filter2 -> SetInput (filter1 -> GetOutput());
```

A saída do ‘filter1’ é conectada na entrada do ‘filter2’. Se a saída do ‘filter1’ não for compatível com a entrada do ‘filter2’ ocorrerá um erro. As duas maneiras de resolver isso foram apresentadas no texto anterior sobre o *pipeline*.

Alguns filtros admitem mais de uma entrada. Múltiplas saídas são comuns.

Na verdade a maioria dos filtros é implementada com uma única saída, mas a multiplicidade é conseguida permitindo acesso à saída de um objeto por vários outros objetos, como no exemplo das quádricas, onde o filtro de *outline*, o de contorno, e o de extração de geometria liam a única saída do filtro de amostragem (*Sample filter*).

Alguns objetos têm efetivamente múltiplas saídas, como, por exemplo, o processo `vtkExtractVetorComponents` que toma um vetor e devolve o valor escalar de seus componentes x, y e z.

Objetos que efetivamente admitem mais de uma entrada (como p. ex. o filtro `vtkGlyph3D`), possuem métodos diferentes para setar entradas diferentes, como o objeto `vtkGlyph3D`.

Exemplo:

```
glyph = new vtkGlyph3D;
glyph->SetInput(foo->GetOutput()); // uma entrada
glyph->SetSource(bar->GetOutput()); // outra entrada
```

`vtkGlyph3D` copia a geometria definida em *SetSource* para cada ponto definido por *SetInput*.

Vários outros objetos admitem multiplicidade efetiva de entradas e saídas.

5.4.3 LOOPS

VTK admite *looping*, ou seja, que a saída do objeto seja realimentada na entrada de um objeto anterior a ele no *pipeline*, para afetar sua própria entrada. Duplo *rendering* é necessário para efetivar essa influência (veja exemplo abaixo) [SCH96].

Esfera com mapeamento de cor para distância ao plano.

```
#------------------------------------------------------------------
# ColorSph.cpp
# Esfera com mapeamento de cor para distância ao plano.
#include "vtk.h"

main ()
{
      vtkRenderer *renderer = vtkRenderer::New();
      vtkRenderWindow *renWin = vtkRenderWindow::New();
            renWin->AddRenderer(renderer);
      vtkRenderWindowInteractor *iren=vtkRenderWindowInteractor::New();
            iren->SetRenderWindow(renWin);

      vtkSphereSource *sphere = vtkSphereSource::New();
            sphere->SetPhiResolution(12); sphere->SetThetaResolution(12);

      vtkElevationFilter *colorIt = vtkElevationFilter::New();
            colorIt->SetInput(sphere->GetOutput());
            colorIt->SetLowPoint(0,0,-1);
            colorIt->SetHighPoint(0,0,1);

      vtkDataSetMapper *mapper = vtkDataSetMapper::New();
            mapper->SetInput(colorIt->GetOutput());

      vtkActor *actor = vtkActor::New();
            actor->SetMapper(mapper);

      renderer->AddActor(actor);
      renderer->SetBackground(1,1,1);
      renWin->SetSize(450,450);

      renWin->Render();
      iren->Start();
      renderer->Delete();
      renWin->Delete();
      iren->Delete();
      sphere->Delete();
      colorIt->Delete();
      mapper->Delete();
      actor->Delete();
}
#------------------------------------------------------------------
```

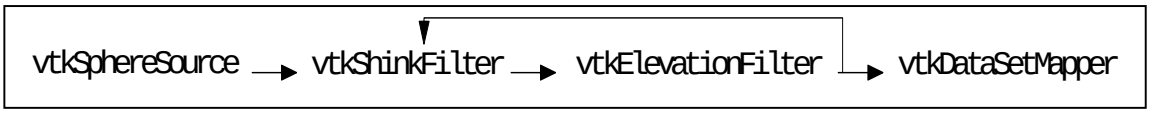
O exemplo acima é um exemplo de um *pipeline* simples. O *pipeline* de visualização inclui um mapeamento de uma função implícita (esfera) diretamente num filtro de elevação. Esse filtro calcula a distância de todos os pontos a uma linha (ou plano) específicos. No exemplo, `vtkElevationFilter` calcula a distância de todos os pontos

da esfera (com a resolução especificada, no caso 12) para um plano paralelo ao plano xy, passando pelo ponto (0,0,-1). A saída desse filtro inclui as distâncias calculadas mais a geometria por trás dos dados originais (a esfera).

Em seguida esses dados são passados para um mapeador (sempre o último passo no *pipeline* de visualização antes do início do *pipeline* de renderização). Esse Mapeador (`vtkDataSetMapper`), mapeia para cor os valores recebidos. Se a tabela de cores (*LookUpTable*) não for definida, ele usa a *rainbow* (arco-íris).

O *pipeline* de visualização é (Quadro 13).

Quadro 13 - *Pipeline* de visualização



Depois disso, tem-se a parte usual de renderização, ou seja, a definição dos objetos de renderização (RenderMaster, RenderWindow e Renderer e Interactor), a definição dos atores para o renderizador, e renderização + interação. Atores são sempre associados a resultados de mapeadores.

5.5 TRABALHOS CORRELATOS

Podemos citar alguns trabalhos correlatos, como “Visualização de Dados Geológicos” da acadêmica Karen Basso da Universidade Federal do Rio Grande do Sul, onde ela ocupa uma ferramenta de visualização de imagens usando o VTK para a criar imagens que mostram capas geológicas como superfícies tridimensionais [BAS98]. Também o artigo sobre “Visualização e Acompanhamento Automático de Sistemas de Nuvens” escrito por [SAN96] onde apresenta técnicas de aplicação meteorológicas e alguns aspectos e discussões sobre técnicas de visualização cientifica [SAN96] e por último os exemplos de aplicações do VTK ilustrados por [SCH96] em seu livro.

6 DESENVOLVIMENTO

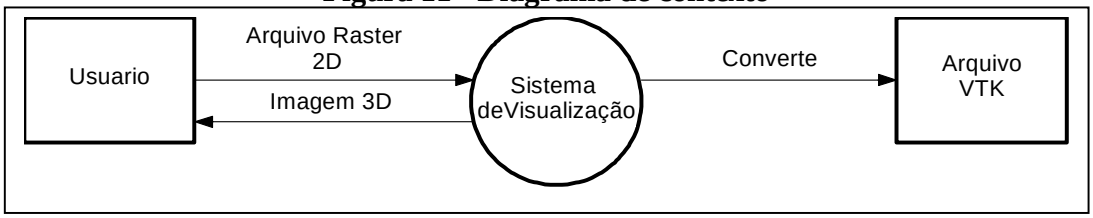
6.1 ESPECIFICAÇÃO

O protótipo foi desenvolvido na linguagem de programação Visual C++ 6.0 da Microsoft com a incorporação da biblioteca do VTK versão 2.01. O padrão do VTK para vizualização dos dados é baseado no sistema de coordenadas dado pela regra da Mão-Direita descritos no capítulo 3. Para implementação da obtenção das amostras utilizou-se de estrutura de dados do tipo lista encadeada duplamente com alocação dinâmica de memória (ver anexo 1).

6.1.1 DIAGRAMA DE CONTEXTO

Segue a abaixo (ver figura 11) o diagrama de contexto conforme o protótipo.

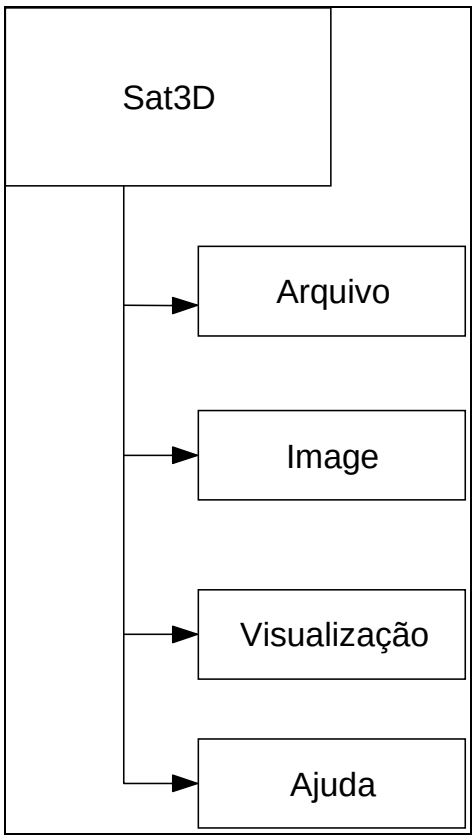
Figura 11 - Diagrama de contexto



6.1.3 DIAGRAMA HIERÁRQUICO FUNCIONAL

O protótipo desenvolvido chamado de Sat3D encontra-se assim definido dentro do diagrama hierárquico funcional (ver figura 12).

Figura 12 - DHF SAT3D



61.4 - FORMATO DO ARQUIVO VTK

Os formatos de arquivo do VTK consistem em cinco partes básicas (ver Quadro 14):

- A primeira parte é a versão do arquivo e identifacador. Esta parte contém uma única linha: `#vtkDataFile Version x.x.` Esta linha deve ser

exatamente como está mostrado com a exceção do número da versão representado pelo `x.x` que variará com lançamentos diferentes de VTK. (a versão atual do número é 2.0. Versão 1.0 arquivos são compatíveis com versão 2.0 arquivos.).

- A segunda parte é o *header*. O *header* consiste em uma cadeia de caráter terminado pelo caracter *end-of-line*. O *header* é de no máximo256 caráteres. O *header* podem ser usados para descrever os dados e inclui qualquer outra informação pertinente.
- A próxima parte é o formato de arquivo. O formato de arquivo descreve o tipo de arquivo, ASCII ou binário. Nesta linha a única palavra que tem que aparecer é ASCII ou BINARY.
- A quarta parte é a estrutura do `dataset`. A parte de geometria descreve a geometria e topologia do `dataset`. Esta parte começa com uma linha que contém a palavra chave `DATASET` seguida por outra palavra chave que descreve o tipo de `dataset`.
- A parte final descreve os atributos de `dataset`. Esta parte começa com a palavra chave `POINT_DATA` ou `CELL_DATA`, seguidos por um número de inteiro que especifica o número de pontos ou células, respectivamente. (Não importa se POINT_DATA ou CELL_DATA vem primeiro.) Outras combinações de palavras chaves definem então o `dataset` atual que atribuem valores (i.e.,escalas, vetores que textura coordena, ou dados de campo).

Quadro 14 - Estrutura do arquivo VTK

```
#vtkDataFile Version 2.0  ] (1)
Exemplo                   ] (2)
ASCII | BINARY            ] (3)
DATASET type     ---------|
...              ---------| (4)
POINT_DATA n     ---------+
...                       | (5)
CELL_DATA n               |
...              ---------+
```

6.1.5 TRIANGULARIZAÇÃO

Para a representação tridimensional das nuvens foi necessário a utilização de triangularização (um conjunto de pontos), e a técnica que melhor se apresentou para

solução do problema foi a técnica de geração da grade regular retangular que apresentaremos a seguir.

6.1.6 GRADE REGULAR RETANGULAR

Uma das principais alternativas das grades irregulares triangulares é a grade regular retangular, já que estas estruturas possuem duas principais formas de representação, as grades irregulares triangularizadas e as grades retangulares [REI97].

As grades irregulares triangularizadas são basicamente geradas pelos algoritmos de triangularização, que consistem de um conjunto de triângulos não sobrepostos cujas uniões cobrem totalmente a região mas não se estendem à cobertura convexa.

A grade regular retangular representa a superfície como uma distribuição regular do conjunto de pontos com as elevações armazenadas em um vetor. Desta forma para obter as coordenadas x, y e z da grade necessita-se ter os valores iniciais das coordenadas x e y, os intervalos entre estas duas coordenadas e pesquisar seqüencialmente o vetor de elevações para definir a coordenada z. Cada representação possui suas vantagens e desvantagens. A escolha de uma representação em relação a outra depende do tipo de aplicação em que o modelo é utilizado [REI97].

Entre as vantagens de um modelo de grades regulares retangular, encontram-se [REI97]:

- a posição espacial de cada ponto pode ser calculado através da referência geográfica e a resolução das coordenadas (xy) na grade, previamente armazenadas;
- é a estrutura de dados mais utilizada por causa de sua facilidade de implementação e eficiência computacional e
- a facilidade e sofisticação para exibir superfícies, onde perspectivas de uma posição se constituem de seções paralelas através da grade produzindo impressões realísticas de terrenos complexos. Exibições matriciais de declividade ou elevação são facilmente geradas por causa da congruência no formato dos dados e do equipamento de exibição.

Entre as suas desvantagens têm-se [REI97]:

- os pontos de superfície são formados por pontos interpolados, a menos que estes estejam localizados exatamente nos vértices da grade retangular;
- não se pode facilmente manipular alterações abruptas nas elevações, não permitindo superfícies verticais ou muito próximas a estas;
- fronteiras irregulares ao redor do espaço de dados ou buracos internos são difíceis de serem definidos;
- operações matemáticas são computacionalmente difíceis e ambíguas, já que o tamanho da malha da grade afeta os resultados obtidos e a eficiência computacional;
- os caminhos de fluxo de declividade usados nas análises (hidrológicas) tendem a gerar trajetórias em ziguezague, tornando, portanto, esta análise um tanto imprecisa e não realística, e
- os pontos devem ser ajustados à rugosidade do terreno devendo-se definir uma resolução com um intervalo menor, criando-se assim redundância nos pontos em superfícies planas.

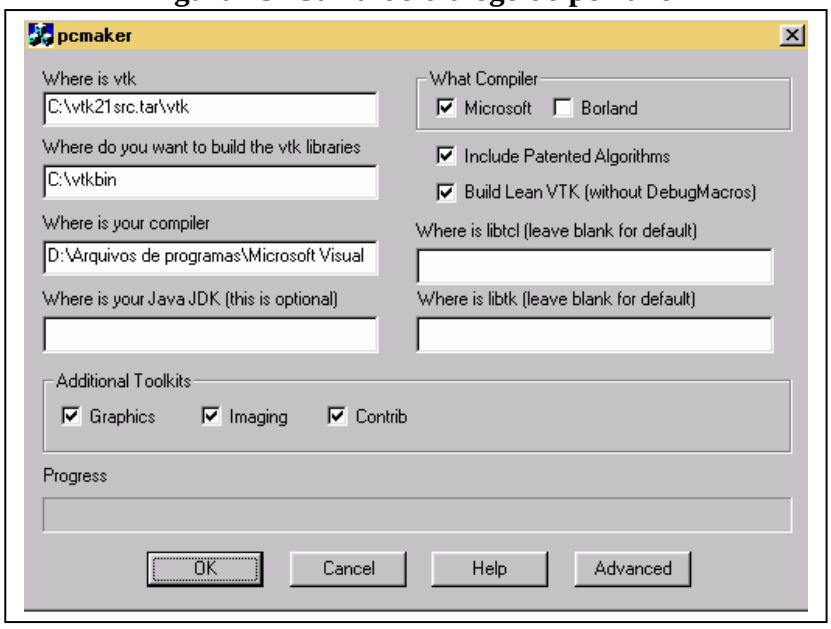
Em resumo, para algumas aplicações, como representação de superfícies as grades regulares retangulares possuem uma expressiva vantagem em relação as outras técnicas de triangularização [REI97]. Por este e outros motivos optou-se por usar a grade regular retangular para desenvolver-se os filtros que fazem a conversão dos arquivos.

6.1.8 COMPILAR A VERSÃO 2.01 DO VTK

Para usar o VTK com o Visual C++ é necessário compilar a versão 2.0 onde será gerado a biblioteca do VTK (*vtkdll.lib*), segue-se os passos descritos abaixo:

- Primeiro deve-se executar o programa *pcmaker.exe* que é encontrado no subdiretótio *pcmaker/Debug/* do diretório onde foi instalado o VTK. Ao ser executado o *pcmaker* abre uma caixa-de-diálogo (ver figura 13) com as opções de compilação, como por exemplo o tipo do compilador (Microsof ou Borland) deverá ser usado, e quais partes do VTK deverão ser incluídas (*Graphics*, *Imaging*, *Contrib*).
- Executar o *nmake* após completada a execução do *pcmaker* (Microsoft Compiler) ou o make (Borland Compiler) a partir dos diretórios vtkbin/vtkdll, para gerar respectivamente as DLLs vtkdll.dll. Esta operação é demorada e requer 250 MB de disco rígido.

Figura 13- Caixa-de-diálogo do pcmaker



Desta forma tem-se a biblioteca Vtkdll.lib ao qual deverá ser incluida no projeto onde deseja-se utilizar as funções do VTK.

6.2 SISTEMA

O Sistema de Visualização Cientifica (SAT3D) foi desenvolvido como protótipo para trabalhar com imagens de satélites (arquivos de imagens nos formatos BMP, GIF, JPEG E PNG). A princípio estas imagens são obtidas dos satélites Goes disponibilizadas na *Internet*, mas podem ser usadas imagens de outros satélites ou outros locais de aquisição, desde que as imagens sejam no formato BMP, GIF, JPEG e PNG.

O SAT3D possui as funções básicas de um sistema aplicativo, bem como funções de obtenção e alteração de uma imagem, funções de controle de janelas, funções de exportação para o formato VTK e visualização em ambiente de visualização científica.

O SAT3D permite trabalhar simultaneamente com diversos arquivos de imagens abertos, pois trabalha com os recursos de MDI (*Multiple Document Interface*) do ambiente de desenvolvimento do Visual C++. Todas as funções do SAT3D referenciam sempre o arquivo da janela ativa.

6.3 FUNÇÕES BÁSICAS DO SISTEMA

A janela principal do sistema possui um menu Principal, uma *SpeedBar*, uma barra de S*tatus* e uma barra de Ajuda, como visto na figura 14.

O menu Principal é composto pelos itens de menu Arquivo, Imagem, Visualização e, Ajuda o qual encontra-se na parte superior da janela principal do sistema.

O *SpeedBar* (figura 15) é uma barra de ícones localizada logo abaixo do menu Principal. Estes ícones funcionam como atalho para as funções mais utilizadas, sendo mais uma opção de acesso as funções do sistema.

A barra de S*tatus* fica na parte inferior da janela principal logo acima da barra de Ajuda. Esta barra permite visualizar informações referentes a posição do *mouse* sobre um arquivo de imagem aberto.

A barra de Ajuda é encontrada na parte inferior da janela principal onde é fornecida informações do sistema.

Figura 14 - Janela Principal

Figura 15 - *SpeedBar*

6.3.1 ITEM DE MENU ARQUIVO

O SAT3D possui em sua janela principal no item de menu Arquivo do menu Principal opções básicas de tratamento de arquivos tais como Abrir, Fechar e Sair (figura 16).

Para trabalhar-se com o SAT3D é necessário abrir um arquivo de imagens (no formato BMP, GIF, JPEG ou PNG), seja através de uma tecla de atalho Ctrl-A ou pela opção Abrir. Com o arquivo aberto é possível utilizar qualquer função do SAT3D. Ao abrir-se um arquivo de imagem é alocado um espaço na memória para a janela MDI que receberá o arquivo de imagem.

Figura 16 – Item de menu de Arquivo

6.3.2 ITEM DE MENU IMAGEM

No item de menu Imagem (figura 17) pode-se utilizar os recursos de Propriedades que ativa um dialogo onde pode-se configurar o linear, Intervalos e intervalos da amostra (figura 18). A opção de triangularização regular gera um arquivo (TrianReg.g), a opção de Grades – Pontos/Linhas gera um arquivo (GrPonLin.vtk), opção Linhas Nuvem também gera um arquivo (GrPonNuv.vtk). A opção pontos triangularização gera um arquivo de pontos para amostragem das nuvens.

Figura 17 - Item de menu Imagem

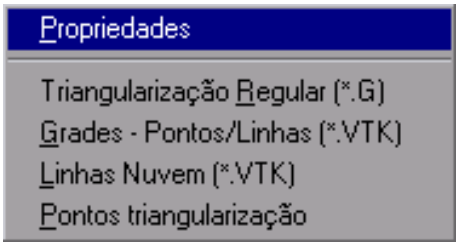
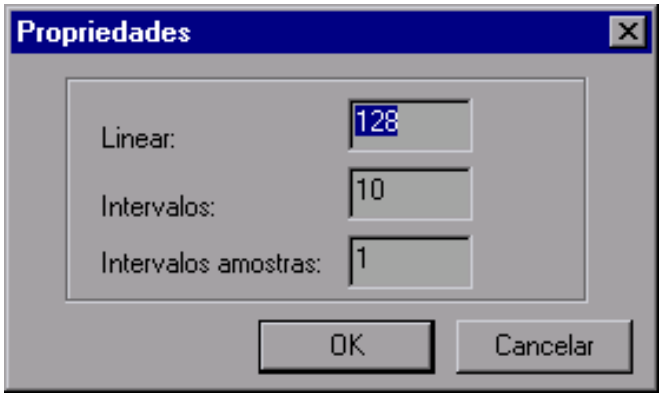


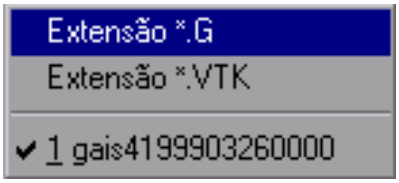
Figura 18 - Item de menu Propriedades



6.3.2 ITEM DE MENU VISUALIZAÇÃO

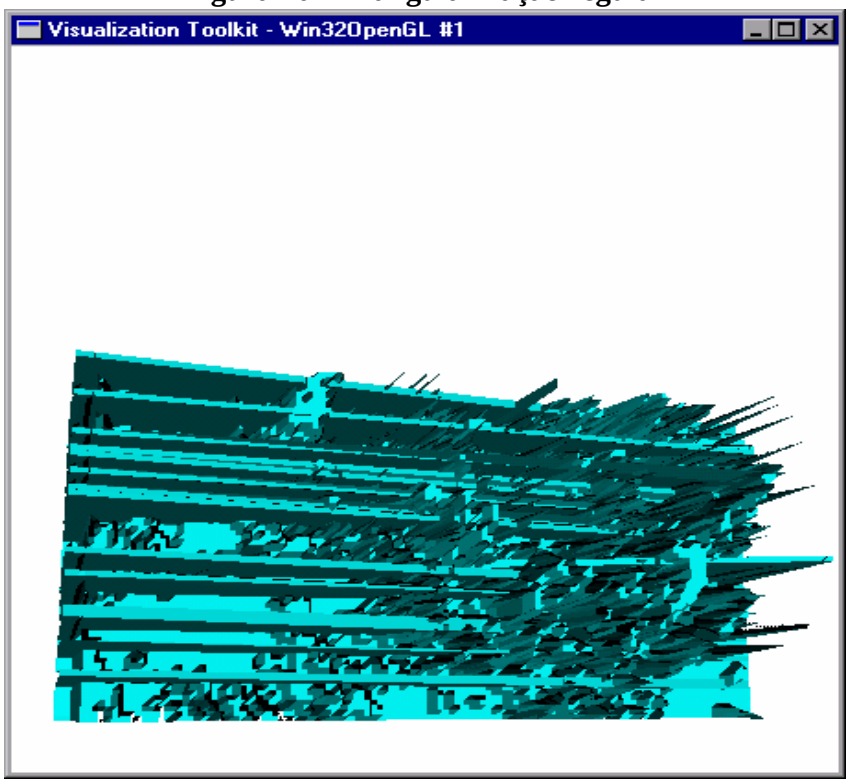
Neste item de menu de visualização (figura 19) temos duas opções a visualização da extensão *.G e da extensão *.VTK, nestas opções é ativado o ambiente de visualização do VTK. Para cada opção deve-se escolher o arquivo gerado pelo opção do menu de imagem. Neste item de menu contém o nome dos arquivos abertos.

Figura 19 - Item de menu Visualização



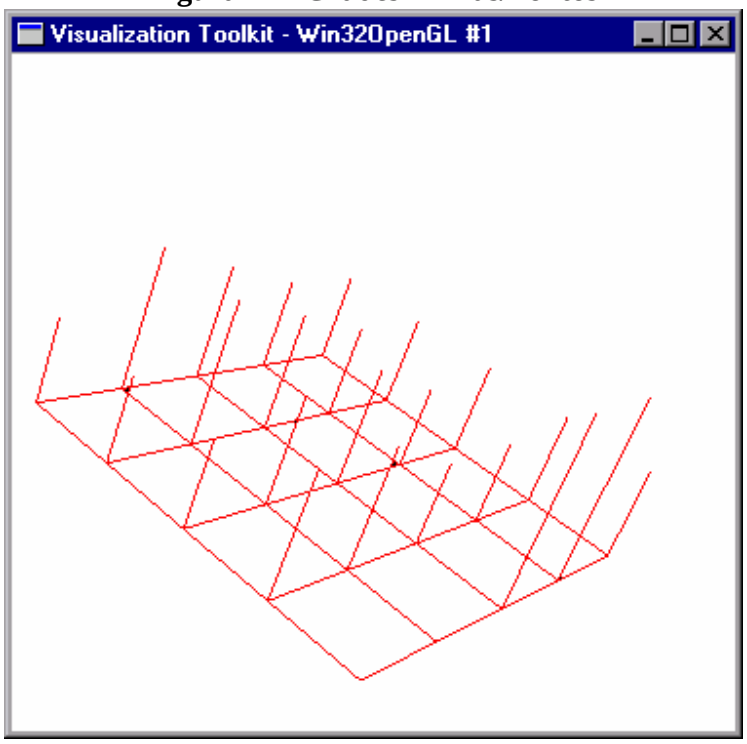
A figura 20 mostra uma imagem gerada pelo VTK que representa o arquivo gerado (TrianReg.g) utilizando-se da técnica de triangularização regular.

Figura 20- Triangularização regular



A figura 21 mostra outra imagem gerada pelo VTK que representa o arquivo (GrPonLin.vtk) onde apresenta a seqüência de formação de uma nuvem e seus dados encapsulados de acordo com os valores do *pixel* de uma imagem. A representação da superfície serve apenas para ilustrar a geração do sólido, pois os *pixels* estão tão próximos que o traçado da superfícies torna-se desnecessário, a representação das linhas verticais é feita através da definição dos intervalos que é definida no menu de item Propriedades (figura 18).

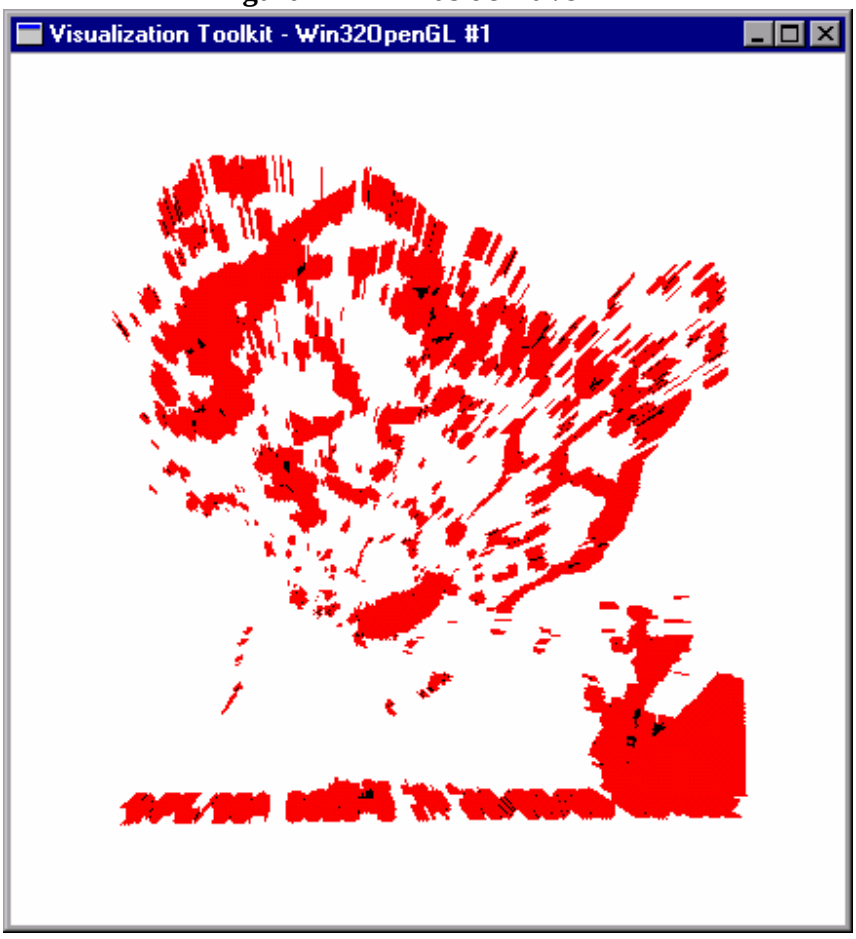
Figura 21 - Grades Linhas/Pontos



A figura 22 mostra outra imagem gerada pelo VTK que representa o arquivo (GrPonNuv.vtk) a técnica usada foi de projetar valores escalares bidimensionais definidos no menu de item propriedades (figura 18) no campo linear (que representa a temperatura de brilho da imagem). Cada valor é associado a uma altura, que está relacionado a uma cor dentro de 256 níveis de cinza (de 0 a 255). Para cada um dos pontos da nuvem são traçados segmentos de retas verticais, cuja cor é associada ao nível de cinza do *pixel* da imagem.

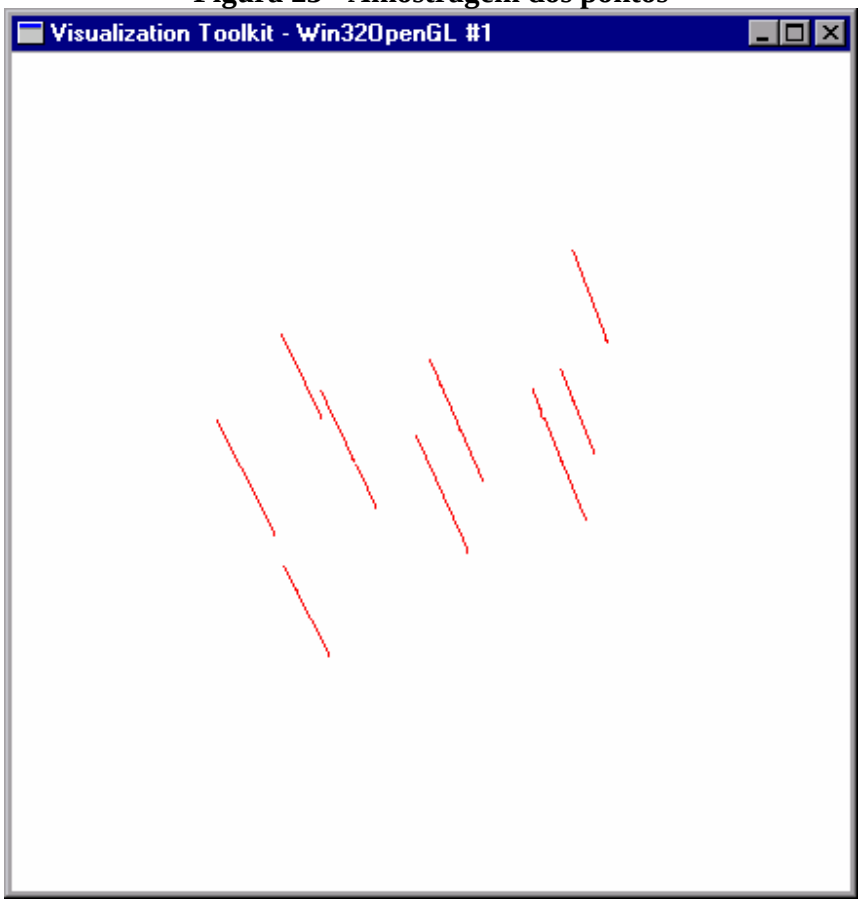
Devido à proximidade dos *pixels* da imagem, o conjunto de segmentos de retas verticais confere um aspecto de solidez tridimensional à nuvem. Com esta informação é possível observar a provável forma de cobertura da nuvem e ainda identificar os topos das mesmas. Os topos, identificados com uma cor mais branca, indicam os pontos mais frios da nuvem.

Figura 22 - Linhas de Nuvem



A figura 23 mostra a visualização da trajetória do centróide sistema. O centróide é calculado através da médias dos pontos em x e y representados pelos seus vizinhos que constituem uma nuvem. Sua representação é feita através da conexão dos centróides do sistema, calculados em cada imagem. Esta técnica permite inferir um comportamento temporal observando-se apenas a imagem tridimensional gerada. O algoritmo gerado utiliza-se dos valores gerados definidos no item de menu Propriedades (ver figura 18).

Figura 23 - Amostragem dos pontos



7 CONCLUSÕES E EXTENSÕES

7.1 CONCLUSÕES

O presente trabalho apresenta os resultados do estudo sobre Visualização Científica aplicada à Meteorologia, onde foram usadas imagens de satélite (2D). Utilizou-se do conceito de triangularização, onde utilizou-se a estrutura dos Triângulos Regulares Retangulares, para representação tridimensional das nuvens sobre a topografia da região.

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

- carregar arquivos raster 2D (imagens de satélite);
- converter o arquivo raster 2D para o formato de arquivo do VTK;
- ser capaz de representar em 3D a imagem raster;
- possibilitar a manipulação da imagem 3D em um ambiente de câmera sintética
- extrair amostras representativas das regiões de interesse (nuvens).

O resultado mais importante foi a geração das amostras, pois sabe-se que são valores imprescindíveis [SAN96] para um estudo meteorológico através de uma seqüência de imagens.

Este protótipo também permite dar continuidade aos objetivos iniciais através das propostas de extensões, permitindo ter-se outros produtos que auxiliem à análise meteorológica em centros de pesquisa.

7.3 LIMITAÇÕES

Ao fechar a janela de visualização do VTK as funções internas do VTK estão fechando todo sistema.

A falta de pré-filtragem para retirada de “ruídos” (legenda, divisão política, etc.) da imagem.

7.3 EXTENSÕES

Como extensão ao protótipo pode-se ainda desenvolver a visualização utilizando de animações de nuvens para um melhor acompanhamento e visualização da mesma.

As funções de obtenção e alteração de informações da imagem realizam uma varredura *pixel* a *pixel* no arquivo de imagens. Esta varredura *pixel* a *pixel* utiliza um grande tempo esta demora poderia ser minimizada através das classes do VTK e realização de um estudo sobre os diversos tipos de formatos do VTK para a representação da imagem em 3D.

Sugere-se um estudo de técnicas para a visualização dos pontos das amostras para a representações das nuvens.

No ambiente de visualização do VTK está apresentando um pequeno problema, ao sair do ambiente de visualização o mesmo está finalizando toda a aplicação, sugere-se então um estudo no ambiente do VTK para contornar o problema, sabe-se que ocorre quando a opção de interação de imagens com o mouse é ativada.

Outro recurso que o protótipo poderia apresentar é a aquisição de imagens raster (imagens de satélite), através de uma opção que mostrasse *links* de endereços de internet onde pudesse obter as imagens de satélite.

ANEXO

Anexo 1 - Programa amostragem dos dados

```
// definição das estruturas para implementação das lista encadeadas duplamente
struct Pontos
{
	int pos_x;
	int pos_y;
	int pos_z;
	Pontos *proximo;
	Pontos *anterior;
};

struct Pontos_Vizinhos
{
	int pos_x;
	int pos_y;
	int pos_z;
	BOOL visitado;
	Pontos_Vizinhos *anterior;
	Pontos_Vizinhos *proximo;
};

struct Faixa
{
	int valor_faixa;
	int valor_maximo;
	Faixa *anterior_faixa;
	Pontos *aponta_pontos;
	BOOL Ligado;
};

int intervalos;
int linear;
int amostra;
Faixa *Fim_faixa;
Pontos *Fim_pontos;
Pontos *Prim_novos;
Pontos *novospontos;
Pontos *pontos_triangularizacao;
Pontos *Fim_triang;
Pontos_Vizinhos *pontos_viz;
Pontos_Vizinhos *Fim_viz;
Pontos_Vizinhos *Prim_viz;
Faixa *novafaixa;

// Função Principal
void CSat3dView::OnImagePontostriangularizao()
{
Desaloca(); // Se tiver alguma estrutura alocada então desaloca
for(int x=linear;x<=255;x+=amostra)
{
	novafaixa = new Faixa;
	novafaixa->valor_faixa=x;
	novafaixa->valor_maximo= x + amostra - 1;
	novafaixa->Ligado=FALSE;
	novafaixa->anterior_faixa=Fim_faixa;
	Fim_faixa=novafaixa;
}
// Define o tamanho da imagem selecionada
int Width = GetDocument()->GetImage()->GetWidth();
int Height = GetDocument()->GetImage()->GetHeight();
```

```
COLORREF corb;
PALETTEENTRY cores;
DWORD valor=(DWORD)linear;
void *ponteiro_faixa=NULL;
CProgressDlg dlg;
dlg.Create(this);
dlg.SetRange(0, Width);

for( x=0;x<=Width;x++)
{
	dlg.SetPos(x);
	for(int y=0;y<=Height;y++)
	{

		GetDocument()->image->
GetRGB(x, y, &cores.peRed,&cores.peGreen,&cores.peBlue);
		corb = GetRValue(cores.peRed);
		if(corb>=valor)
		{
			ponteiro_faixa=Busca_Ponteiro_Faixa(corb);
			if(ponteiro_faixa!=NULL)
			{
				novospontos = new Pontos;
					if(Prim_viz==NULL)

				if(Prim_novos==NULL)
					Prim_novos = novospontos;
				else
					Fim_pontos->proximo=novospontos;
				novospontos->pos_x = x;
				novospontos->pos_y = y;
				novospontos->pos_z = corb;
				novospontos->anterior = Fim_pontos;
				novospontos->proximo  = NULL;
				Fim_pontos = novospontos;
				// seta o ponteiro de pontos com faixa
				novafaixa = (Faixa*)ponteiro_faixa;
				if(!novafaixa->Ligado)
				{
					novafaixa->aponta_pontos = novospontos;
					novafaixa->Ligado=TRUE;
				}
			}
		}

	}
}
// Mostra um dialogo com o progresso da tarefa
dlg.SetPos(0);
dlg.SetRange(0, 100);
dlg.SetPos(50);
void *atualpontos = NULL;
while(novospontos!=NULL)
{
	InserePontosVizinhos(novospontos->pos_x,novospontos->pos_y,
 novospontos->pos_z,TRUE);
	Retira_NovosPontos();
	BuscaVizinhos();
	Pontos_Vizinhos *novoponto;

	void * ponteiro=CalculaPontoMedio();
	novoponto = (Pontos_Vizinhos*)ponteiro;
	InserePontosTriang(novoponto->pos_x,novoponto->pos_y, novoponto->pos_z);
	Retira_PontosViz();
	Fim_viz = NULL;
	Prim_viz=NULL;
}
```

```
char* lpTempArq;
CString sString="Pontri.txt";
lpTempArq = sString.GetBuffer(sString.GetLength( ));
sString.ReleaseBuffer( );

CStdioFile Arquivo;
Arquivo.Open(lpTempArq,CFile::typeText | CFile::modeCreate | CFile::modeWrite);
void *atualtriang = NULL;
char pos_x[20], pos_y[20],pos_z[20];
CString Palavra;
dlg.SetPos(75);
while(pontos_triangularizacao!=NULL)
{
	_itoa(pontos_triangularizacao->pos_x,pos_x,10);
	_itoa(pontos_triangularizacao->pos_y,pos_y,10);
	_itoa(pontos_triangularizacao->pos_z,pos_z,10);
	Palavra = pos_x;
	Palavra+=" , ";
	Palavra+=pos_y;
	Palavra+=" , ";
	Palavra+=pos_z;
	Arquivo.WriteString(_T(Palavra));
	Arquivo.WriteString(_T("\n"));
	atualtriang=pontos_triangularizacao->anterior;
	delete(pontos_triangularizacao);
	pontos_triangularizacao=(Pontos*)atualtriang;
}
Arquivo.Close();
dlg.SetPos(100);

AfxMessageBox("Processo de triangularização terminado");

}

void *CSat3dView::Busca_Ponteiro_Faixa(int valor_faixa)
{
Faixa *atual=Fim_faixa;
while(atual!=NULL)
{
	if(valor_faixa >=atual->valor_faixa&&valor_faixa<=atual->valor_maximo)
	{
		return atual;
	}
	atual = atual->anterior_faixa;
}
return NULL;
}

void CSat3dView::Desaloca()
{
if(Fim_faixa!=NULL)
{
	void *atualfaixa = NULL;
	while(novafaixa!=NULL)
	{
		atualfaixa=novafaixa->anterior_faixa;
		delete(novafaixa);
		novafaixa=(Faixa*)atualfaixa;
	}
}
if(Fim_pontos!=NULL)
{
	void *atualpontos = NULL;
	while(novospontos!=NULL)
	{
		atualpontos=novospontos->anterior;
		delete(novospontos);
		novospontos=(Pontos*)atualpontos;
```

```
        }
}
}

void CSat3dView::Retira_NovosPontos()
{
 void *atualpontos = NULL;
 void *atualproximo = NULL;

 atualpontos=novospontos->anterior;
 atualproximo=novospontos->proximo;
 if(atualpontos==NULL&&atualproximo==NULL)
 {
        delete(novospontos);
        novospontos = NULL;
        Fim_pontos = NULL;
        Prim_novos = NULL;
        return;
 }
 if(novospontos==Prim_novos)
 {
        delete(novospontos);
        novospontos = (Pontos*)atualproximo;
        novospontos->anterior=NULL;
        Prim_novos = novospontos;
        return;
 }

 if(novospontos==Fim_pontos)
 {
        delete(novospontos);
        novospontos = (Pontos*)atualpontos;
        novospontos->proximo = NULL;
        Fim_pontos = novospontos;
        return;
 }
delete(novospontos);
 novospontos = (Pontos*)atualpontos;
 novospontos->proximo = (Pontos*)atualproximo;

 novospontos = (Pontos*)atualproximo;
 novospontos->anterior = (Pontos*)atualpontos;
 novospontos= Fim_pontos;
}

BOOL CSat3dView::Pesquisar_Vizinhos(int x, int y)
{
 BOOL achou=FALSE;
 Pontos *atual = novospontos;
 while(atual != NULL)
 {
       if(atual->pos_x==x&&atual->pos_y==y)
   {
          achou=TRUE;
          novospontos=atual;
          break;
   }
   atual = atual->anterior;
 }

 return achou;
}

void CSat3dView::InserePontosVizinhos(int x, int y, int z, BOOL visitado)
{
 pontos_viz = new Pontos_Vizinhos;
 if(Prim_viz==NULL)
        Prim_viz = pontos_viz;
```

```
else
	Fim_viz->proximo=pontos_viz;

pontos_viz->pos_x = x;
pontos_viz->pos_y = y;
pontos_viz->pos_z = z;
pontos_viz->visitado = visitado;
pontos_viz->anterior = Fim_viz;
pontos_viz->proximo=NULL;
Fim_viz = pontos_viz;

}

void CSat3dView::InserePontosTriang(int x, int y, int z)
{
pontos_triangularizacao = new Pontos;
pontos_triangularizacao->pos_x = x;
pontos_triangularizacao->pos_y = y;
pontos_triangularizacao->pos_z = z;
pontos_triangularizacao->anterior = Fim_triang;
Fim_triang = pontos_triangularizacao;
}

void* CSat3dView::CalculaPontoMedio()
{

void *pontoatual;
BOOL achou=FALSE;
Pontos_Vizinhos *atual = Fim_viz;
int contador=0;
int somatorio_x=0, somatorio_y=0;
int razao_x=0, razao_y=0;
while(atual != NULL)
{
	somatorio_x+=atual->pos_x;
	somatorio_y+=atual->pos_y;
	contador++;
	atual = atual->anterior;
}
if(somatorio_x>0&&contador>0)
	razao_x = somatorio_x / contador;
if(somatorio_y>0&&contador>0)
	razao_y = somatorio_y / contador;

atual = Fim_viz;

int d1=pow((razao_x - atual->pos_x),2) + pow((razao_y - atual->pos_y),2);
int d2=0;
pontoatual = atual;
atual = atual->anterior;
while(atual!=NULL)
{
	int d2=pow((razao_x - atual->pos_x),2) + pow((razao_y - atual->pos_y),2);
	if(d2<d1)
	{
		pontoatual=atual;
		d1=d2;
	}
	atual = atual->anterior;
}
return pontoatual;
}

void CSat3dView::Retira_PontosViz()
{
if(Fim_viz!=NULL)
{
	void *atualviz = NULL;
```

```
        while(pontos_viz!=NULL)
        {
                atualviz=pontos_viz->anterior;
                delete(pontos_viz);
                pontos_viz=(Pontos_Vizinhos*)atualviz;
        }
}
}

void CSat3dView::BuscaVizinhos()
{
int x1=0, x2=0, y1=0, y2=0;
x1 = pontos_viz->pos_x+1;
x2 = pontos_viz->pos_x-1;
y1 = pontos_viz->pos_y+1;
y2 = pontos_viz->pos_y-1;
BOOL pontoachado=FALSE;
if(Pesquisar_Vizinhos(pontos_viz->pos_x,y2)) ///
{
        pontoachado =TRUE;
        InserePontosVizinhos(pontos_viz->pos_x,y2, pontos_viz->pos_z,FALSE);
        Retira_NovosPontos();
}
if(Pesquisar_Vizinhos(pontos_viz->pos_x,y1))///
{
        pontoachado =TRUE;
        InserePontosVizinhos(pontos_viz->pos_x,y1, pontos_viz->pos_z,FALSE);
        Retira_NovosPontos();
}
if(Pesquisar_Vizinhos(x2,pontos_viz->pos_y))///
{
        pontoachado =TRUE;
        InserePontosVizinhos(x2,pontos_viz->pos_y,pontos_viz->pos_z,FALSE);
        Retira_NovosPontos();
}
if(Pesquisar_Vizinhos(x1,pontos_viz->pos_y))///
{
        pontoachado =TRUE;
        InserePontosVizinhos(x1,pontos_viz->pos_y, pontos_viz->pos_z,FALSE);
        Retira_NovosPontos();
}


if(!pontoachado)
        InserePontosTriang(pontos_viz->pos_x,pontos_viz->pos_y, pontos_viz->pos_z);
else
{
        pontos_viz=Prim_viz;
        Pontos_Vizinhos *Atual_viz=NULL;
        while(TRUE)
        {
                if(!pontos_viz->visitado)
                {
                        pontos_viz->visitado = TRUE;
                        Atual_viz = pontos_viz;
                        BuscaVizinhos();
                        pontos_viz = Atual_viz;
                }
                if(pontos_viz->proximo==NULL)
                        break;
                pontos_viz = pontos_viz->proximo;
        }
 }
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

[BAS98] BASSO, Karen. **Descrição dos Projetos Pessoais.** Porto Alegre, 1998. Endereço eletrônico: http://www.inf.ufrgs.br/~karen/karen.htm.

[BRO96] BROWN, C. Wayne. ***Graphics file formats - Reference and guide***. Estados Unidos: Prentice Hal, 1996.

[CIV76] CIVITA, Victor. Enciclopédia Abril. **Meteorologia,** v. 8, p. 104-106. São Paulo (SP): Abril, 1976.

[CRO93] CRÓSTA, Álvaro P. **Processamento digital de imagens de sensoriamento remoto**. Campinas (SP): UNICAMP, 1993.

[CUL97] CULLENS, Chane. **Usando Visual C++ 4.0**. Rio de Janeiro, 1997.

[DOR97] DOROW, Anatoli, **Desenvolvimento de um sistema de análise Meteorológica – SAM**. Universidade Regional de Blumenau, Blumenau, 1997.

[FAL93] FALCON, Jacques. **Processamento e análise de imagens.** Província de Córdoba – Argentina, 1993.

[GRA91] GRAN, Manoel Alouso. **Satélites Meteorológicos**. Curso de Extensão Universitária - Interpretação de Imagens e Análise Meteorológica, São José dos Campos, 1991.

[KRU97] KRUGLINSKI, David J. ***Inside Visual C++***. Washington - Estados Unidos: Microsoft Press, 1997.

[MER97] MEREGE, Pedro. **Geoprocessamento introdução**. Curitiba (PR): Sagres, 1997.

[PER89] PERSIANO, Ronaldo Cersar Marinho. **Introdução à computação gráfica**: Rio de Janeiro, 1989.

[REI97] REIS, Dalton Solano dos, **Análise da geração de grades irregulares**

triangularizadas em modelos numéricos de terreno. Universidade Federal do Rio Grande do Sul, Porto Alegre, 1997.

[SAN96] SANTOS, Selan Rodrigues Dos. **Visualização e acompanhamento automático de sistemas de nuvens.** Rio de Janeiro, 1996. Endereço eletrônico: http://gama.urisan.tche.br/~cati/artigos/visuali.html.

[SCH96] SCHOREDER, Will. ***The Visualization TollKit – Na Object-Oriented approach to 3D graphics*.** Estados Unidos: Prentice Hall, 1996.

[SMI99] **Satélites Meteorológicos – Imagens infravermelho**. 1999. Endereço eletrônico: http://www.cptec.inpe.br/meteoimages/homep.html.

[TAI99] TAINA, Agma **Transformações geométricas 2D e 3D.** 1999. Endereço eletrônico: http://www.gbdi.icmc.sc.usp.br/documentacao/apostilas.