6
Referências Bibliográficas

1 COHEN-OR, D.; CHRYSANTHOU, Y.; SILVA, C. T. A Survey of Visilibity for Wakthrough Applications. **Proceedings of EUROGRAPHICS '00**, course notes, 2000.

2 LINDSTROM, P.; PASCUCCI, V. Visualization of Large Terrains Made Easy. **Proceedings of IEEE Visualization 2001**, San Diego, California, October, 2001, pp. 363-370.

3 LINDSTROM, P.; PASCUCCI, V. Terrain Simplification Simplified: A General Framework for View-Dependent Out-of-Core Visualization. **IEEE Transactions on Visualization and Computer Graphics**, May 8, 2002.

4 TOLEDO, R. **QuadLod: Uma Estrutura para a Visualização Interativa de Terrenos.** Rio de Janeiro, 2000. Dissertação de Mestrado. Departamento de Informática, PUC-Rio.

5 LINDSTROM, P. et al. Real-Time, Continuous Level of Detail Rendering of Height Fields. **Proceedings of SIGGRAPH '96**, pp.106-118. Aug.1996.

6 HOPPE, H. View-Dependent Refinement of Progressive Meshes. **Proceedings of SIGGRAPH 97**, pp.189-198.Aug.1997.

7 DUCHAINEAU, M. A. et al. ROAMing Terrain: Real-time Optimally Adapting Meshes. **IEEE Visualization '97**, pp.81-88. Nov. 1997.

8 DE BOER, W. H. Fast Rendering Using Geometrical MipMapping. **E-mersion Project**, October 2000. Disponível em: <http://www.connectii.net/emersion>.

9 GÜDÜKBAY, U.; YILMAZ T. Stereoscopic View-Dependent Visualization of Terrain Height Fields. **IEEE Transactions on Visualization and Computer Graphics**, vol. 8, no. 4, October-December 2002.

10 EBERLY, D. **3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics,** Morgan Kaufmann, San Francisco, 2001.

11 **Treinamento e Ensino – Sistemas de Jogos Didáticos.** http://www.cgcfn.mar.mil.br/com_pessoal/secoes/secoesprincipais/cp_sjd.htm. Acesso em: 06 ago. 2003.

12 WOO, M.; NEIDER, J.; DAVIS, T. **OpenGL Programing Guide: The Official Guide to Learning OpenGL**, Version 1.1. Addison-Wesley Developers Press, 1997.

13 **Lua - The Programming Language**. http://www.lua.org. Acesso em: 06 ago. 2003.

14 **Puget Sound. http://www.cc.gatech.edu/projects/large_models/ps.html**. Acesso em: 06 ago. 2003.

15 MONTENEGRO, A. **Investigação de Novos Critérios para Inserção de Pontos em Métodos para Simplificação de Modelos de Terreno através de Refinamento.** Rio de Janeiro, 1997. Dissertação de Mestrado. Departamento de Informática, PUC-Rio.

16 **IM – Access Library to Bitmap Image Files**. http://www.tecgraf.puc-rio.br/im. Acesso em: 06 ago. 2003.

Apêndice A – Descrição da Aplicação SJD-Vis3D

O programa SJD-Vis3D[11], que está sendo desenvolvido pelo laboratório Tecgraf (vinculado ao Departamento de Informática da PUC-Rio) para a Marinha do Brasil, permite realizar vôos sobre um terreno. O algoritmo discutido neste trabalho e, mais especificamente, o módulo Terreno apresentados no apêndice B são utilizados para obter, a cada quadro, a malha do terreno sendo sobrevoado. Além disso, o SJD-Vis3D é implementado em linguagem C++, utiliza a biblioteca gráfica OpenGL[12] e arquivos de descrição feitos na linguagem de *scripts* LUA[12].

No SJD-Vis3D a navegação através do terreno pode ser feita com o *mouse* ou com as setas do teclado. Conforme pode ser visto na Figura 30, o SJD-Vis3D imprime na tela algumas informações importantes. São elas: o número de quadros gerados por segundo (*fps,* do inglês *frames per second*); o número de triângulos desenhados; a velocidade da navegação transformada em quilômetros por hora; as coordenadas x, y e z da câmera no espaço do objeto; o valor do erro tolerável; o tamanho percentual do *frustum* de visão em relação à tela e, finalmente, se as operações *culling* da malha fora da visão e *geomorphing* estão sendo utilizadas ou não.

A tecla *w* permite alternar entre mostrar a malha em *wireframe*, ou seja, com os polígonos sem preenchimento, e a malha tradicional com os polígonos preenchidos. A tecla *t* permite alternar entre exibir a malha com ou sem textura. Já as teclas 1, 2, 3 e 4 permitem, respectivamente, escolher entre: refinamento combinado com *culling* e *geomorphing*; refinamento sem *culling* e com *geomorphing*; refinamento sem *culling* e sem *geomorphing* e, finalmente, refinamento com *culling* e sem *geomorphing*. As teclas 5 e 6 permitem alterar (diminuindo e aumentando) o erro tolerável. Similarmente, as teclas 7 e 8 permitem alterar o *frustum* de visão. Além disso, as teclas *a* e *z* auxiliam na navegação, permitindo subir ou descer a câmera em relação ao eixo z. A tecla *p* permite congelar e descongelar a navegação e a tecla *m* permite liberar e vincular os movimentos do *mouse* à navegação. A tecla *x* permite mostrar uma vista no

plano xy do terreno inteiro e retornar à vista anterior. A tecla *r* possibilita alternar entre a exibição da malha de resolução máxima do terreno e retornar à malha anterior. A tecla *c* faz o algoritmo de refinamento parar de ser chamado. Neste caso, a malha de triângulos pára de ser atualizada, mas os movimentos da câmera continuam liberados. O objetivo, neste caso, é permitir inspecionar a malha gerada.

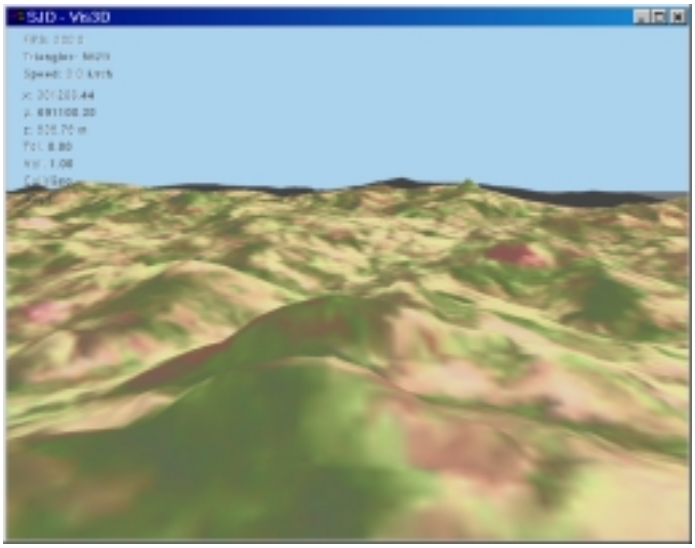

Figura 30 - Janela do SJD-Vis3D.

Conforme já dito, para obter a malha de triângulos aproximada que representa o terreno, o SJD-Vis3D utiliza um módulo Terreno, desenvolvido em paralelo a este trabalho e inspirado no algoritmo proposto por Lindstrom & Pascucci em [2] e estendido em [3]. Na verdade, este módulo implementa os algoritmos listados no capítulo 3. Além disso, a idéia é que ele possa ser utilizado por várias aplicações que necessitem realizar um vôo ou passeio por um terreno ou campo de alturas.

Assim, o programa SJD-Vis3D ou qualquer outro programa que utilizar este módulo fica responsável por implementar tarefas como: leitura dos dados do terreno, ajuste da câmera, interface com o usuário, chamada às funções do módulo Terreno preenchendo adequadamente os parâmetros de entrada e, finalmente, renderização da malha produzida. A fase de renderização da malha foi deixada para a aplicação, em vez de ser implementada no módulo Terreno, simplesmente para dar à aplicação a liberdade de escolher como fazer a aplicação da textura. Em

outras palavras, o intuito foi deixar o módulo o mais genérico possível. O apêndice B descreve este módulo em mais detalhes. Antes disso, porém, apresentamos na próxima seção o formato dos terrenos lidos pelo programa SJD-Vis3D.

A1. Definição dos Terrenos para o SJD-Vis3D

Esta seção descreve como devem ser definidos os terrenos que serão utilizados pela aplicação SJD-Vis3D. Cada terreno é formado por um mapa de alturas e uma textura, sendo que esta última não é obrigatória. A seguir descrevemos o formato dos arquivos de definição do mapa de alturas e depois explicamos os arquivos de textura.

A1.1. Mapa de Alturas

O mapa de alturas é definido por dois arquivos: um de cabeçalho com a extensão .hdr e o outro com extensão .flt contendo os valores das alturas de cada célula. O nome dos dois arquivos difere apenas pela extensão. O formato do mapa de alturas é exatamente o formato binário com que o programa ARC/INFO exporta suas grades a partir de um arquivo-imagem.

O arquivo de cabeçalho (.hdr) é um arquivo texto que descreve as características da grade. O formato deste arquivo é o seguinte:

ncols <número de colunas da grade>
nrows <número de linhas da grade>
xllcorner <coordenada x do canto inferior esquerdo da grade>
yllcorner <coordenada y do canto inferior esquerdo da grade>
cellsize < tamanho de cada célula>
NODATA_value -9999
byteorder LSBFIRST

Os campos '<descrição>' devem ser substituídos pelo valor correspondente. O campo '*NODATA_value*' contém o valor que foi associado às células que na verdade não possuem valor nenhum. Já o campo '*byteorder*' é a ordem com que os *bytes* que compõem os *floats* com o valor de cada altura estão seqüenciados. De

qualquer forma, estas duas linhas não são necessárias, uma vez que o SJD-Vis3D sempre considera que os valores de '*NODATA_value*' e '*byteorder*' são -9999 e LSBFIRST, respectivamente. No momento da execução, o SJD-Vis3D irá considerar que as células sem valor possuem altura igual a zero.

O arquivo de dados (.flt) é um arquivo binário de *floats* (32 bits). Cada *float*, por sua vez, correspondente à altura de cada célula da grade. O primeiro valor da seqüência corresponde ao valor da célula localizada no canto superior esquerdo da grade.

A.1.2 Textura

A textura é uma imagem em qualquer formato aceito pela biblioteca IM[16]. Entre eles podemos citar TIF, JPEG e BMP. Esta imagem pode ser georeferenciada ou não. Caso ela não esteja georeferenciada, a aplicação SJD-Vis3D considera que a imagem se encaixa perfeitamente em cima do terreno.

O georeferenciamento da imagem é feito através de um arquivo no formato *'ESRI worldfile'*, um arquivo-texto com o mesmo nome do arquivo de imagem, mas com a extensão .tfw. O formato deste arquivo é o seguinte:

escala_x
rotação_x
rotação_y
escala_y
x0
y0

Os campos *escala_x* e *escala_y* são respectivamente as escalas horizontal e vertical da imagem. Os campos *x0* e *y0* correspondem às coordenadas do centro do *pixel* de um dos cantos da imagem. Normalmente este *pixel* é o canto superior esquerdo da imagem e por isso a *escala_y* fica negativa. Os campos *rotação_x* e *rotação_y* são respectivamente os ângulos de rotação em x e y da imagem, mas atualmente a aplicação SJD-Vis3D desconsidera qualquer rotação da imagem. As coordenadas reais do centro de cada *pixel* da imagem são então calculadas da seguinte forma:

x = x0 + i * escala_x

y = y0 + j * escala_y

onde:

i é o número da coluna do *pixel* em questão

j é o número da linha do *pixel* em questão

Exemplo:

30.83721033166632

0.00000000000000

0.00000000000000

-30.83721033166632

698365.28008161834000

67343.43848742170700

No exemplo acima pode-se verificar que:

x0 = 698365.28008161834000

y0 = 67343.43848742170700

escala_x = 30.83721033166632

escala_y = -30.83721033166632

rotação_x = 0

rotação_y = 0

Apesar do SJD-Vis3D aceitar qualquer tamanho de imagem, ela só será exibida na resolução máxima aceita pelo OpenGL, que depende da placa de vídeo. Na maioria dos casos essa resolução máxima é de 1024x1024.

Apêndice B – Descrição do Módulo Terreno

Este módulo foi escrito em linguagem C e é composto por duas funções externas, além de outras funções internas. Porém, é importante que os programadores das aplicações que utilizem este módulo necessitem compreender somente estas duas funções externas. Em outros termos, eles precisam saber apenas em que posição dentro da aplicação tais funções devem ser chamadas, quais os parâmetros devem ser passados e quais os parâmetros retornados por elas.

As duas funções externas são: TrnPreProcessamento e TrnGeraMalha. A primeira deve ser chamada uma única vez, na fase de inicialização do programa aplicação. Já a segunda, a função TrnGeraMalha, deve ser chamada toda vez que a aplicação precisar desenhar um novo quadro (por exemplo, toda vez que a câmera se mover). Os parâmetros que cada uma delas recebem e retornam estão descritos detalhadamente no próprio arquivo-cabeçalho do módulo (arquivo terreno.h). Esquematicamente, porém, isto é mostrado no diagrama da Figura 31.

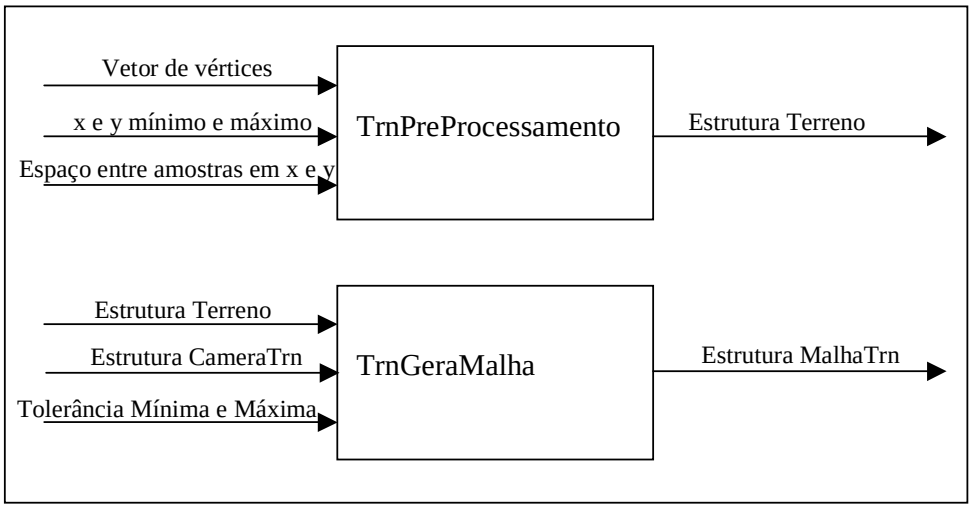

Figura 31 - Diagrama de entradas e saídas das funções TrnPreprocessamento e TrnGeraMalha.

A seguir apresentamos a declaração das principais estruturas de dados utilizadas na interface do módulo Terreno descrito nesta seção.

```
/* Estrutura do vetor de pontos */
Estrutura Vetor de Vértices
{
  float x;
  float y;
  float z;
};

/* Estrutura dos dados de um terreno */
Estrutura Terreno
{
        float xmin;                     /* valor mínimo da coordenada x */
        float ymin;                     /* valor mínimo da coordenada y */
        float deltax;                   /* espaço entre as amostras em x */
        float deltay;                   /* espaço entre as amostras em y */
        VerticeTrn **matrizTer;         /* matriz de vértices do terreno */
        int tamMatriz;                  /* tamanho da matriz de vértices do terreno */
};


/* Estrutura que guarda os parâmetros da câmera necessários  */
Estrutura CameraTrn
{
        float ox, oy, oz;  /* Coordenadas do olho do observador (no espaço do objeto) */
        float largJanela;  /* largura da janela em pixels  */
        float fovx;        /* abertura horizontal da câmera */

        /* Componentes das equações dos 6 planos do frustum de visão */
        float d1, d2, d3, d4, d5, d6;
        float n1x, n1y, n1z; /* Normal unitária do plano 1 */
        float n2x, n2y, n2z; /* Normal unitária do plano 2 */
        float n3x, n3y, n3z; /* Normal unitária do plano 3 */
        float n4x, n4y, n4z; /* Normal unitária do plano 4 */
        float n5x, n5y, n5z; /* Normal unitária do plano 5 */
        float n6x, n6y, n6z; /* Normal unitária do plano 6 */
};

/* Estrutura da malha de saída retornada pelo algoritmo */
Estrutura MalhaTrn
{
        float *Vx;     /* vetor que armazena as coordenadas x dos vértices que deverão
ser desenhados */
        float *Vy;     /* vetor que armazena as coordenadas y dos vértices que deverão
ser desenhados */
        float *Vz;     /* vetor de alturas */
        int tamV;      /* quantidade de vértices presentes na malha final */
};
```

Assim, o parâmetro Vetor de Vértices é, na verdade, um vetor de uma estrutura composta pelas coordenadas x, y e z de cada vértice ou amostra do terreno. Os parâmetros referentes aos valores mínimos e máximos de x e y, respectivamente, e ao espaço entre amostras também nas direções x e y, respectivamente, devem ser calculados previamente pela aplicação e informados como entrada para a função TrnPreProcessamento. A partir daí, esta função

retornará o parâmetro Estrutura Terreno devidamente preenchido e pronto para ser passado como parâmetro de entrada para a função TrnGeraMalha. O programador da aplicação deve lembrar, porém, que é necessário declarar uma variável do tipo da Estrutura Terreno e alocar uma área de memória para ela, antes de chamar a função TrnPreProcessamento.

Além disso, antes de chamar a função TrnGeraMalha, o programador da aplicação que utilizará este módulo deve criar, ainda, uma variável do tipo da estrutura CameraTrn e preencher os seus campos adequadamente. Os parâmetros Tolerância Mínima e Tolerância Máxima, porém, são do tipo real e dizem respeito à faixa de erro para a aproximação da malha gerada. Assim, erros inferiores à Tolerância Mínima serão aceitos e erros que caem na faixa de tolerância serão considerados para o *geomorphing*, conforme já explicado. Finalmente, o programador deve criar e alocar uma área de memória para a estrutura MalhaTrn. É nesta estrutura que serão recebidas as coordenadas a serem renderizadas. No caso de estar utilizando o OpenGL, por exemplo, a renderização basicamente será um comando glBegin (GL_TRIANGLE_STRIP), seguido por um laço com chamadas internas glVertex (com as coordenadas recebidas na estrutura MalhaTrn como parâmetro) e, por fim, o comando glEnd(). O número de iterações deste laço será igual ao número de vértices armazenado na estrutura e também é recebido como parâmetro dentro da estrutura MalhaTrn. Dependendo da escolha feita pela aplicação para a implementação da etapa de aplicação da textura no terreno, o procedimento de renderização, descrito acima, pode ser ligeiramente modificado.