3
Texturização com Dados Geográficos

Neste capítulo, apresentamos aspectos relacionados à geração da textura dos dados geográficos. É apresentado como os dados estão organizados e armazenados, e como essa informação é renderizada utilizando a biblioteca gráfica OpenGL (Shr04). Além disso, as matrizes de transformação e a geração automática de coordenadas para que a textura seja aplicada corretamente ao terreno é mostrada em detalhes.

3.1
Organização dos Dados Geográficos

Os dados geográficos a serem renderizados estão armazenados em um banco de dados no formato Terralib (Cam00). A Terralib é uma biblioteca para auxiliar a construção de aplicativos geográficos, que define um modelo próprio de dados para organizar as informações geográficas. A seguir, temos uma explicação breve dos conceitos relevantes para o desenho da textura dos objetos geográficos.

O conceito inicial para um mapa Terralib é a camada (*layer*). Ela é definida por uma projeção (série de definições sobre como as coordenadas geográficas são projetadas em uma visualização), e um conjunto de objetos geográficos, formado por atributos descritivos e uma representação geométrica, que pode ser ponto, texto, linha ou polígono. Para o presente trabalho, o foco é em representações de linhas e polígonos. Cada representação geométrica dos dados é feita por uma sequência de coordenadas bidimensionais na unidade definida pela projeção.

Para a apresentação visual de uma camada, temos o conceito de tema. Um tema é formado por uma camada, mais informações de estilo visual como cor, espessura da linha, estilo de preenchimento, estilo da linha ou contorno, etc. É possível visualizar apenas um subconjunto dos objetos de uma camada, realizando uma consulta para mostrar apenas os objetos que o usuário considera relevante para aquela visualização. Por exemplo, o usuário dispõe de uma camada de todos os munícipios do Brasil, mas quer um tema que mostre apenas os munícipios do Rio de Janeiro realizando uma consulta adequada. O

tema também permite a visualização de temáticos, um conceito bastante usado em SIGs, onde o estilo do objeto varia de acordo com suas propriedades. Um exemplo clássico é um mapa populacional, onde em um tema de pontos que representam as cidades se utiliza tamanhos maiores no estilo do ponto para representar cidades com maior população.

Por fim, temos o conceito de vista (*view*), que é definida por um conjunto de temas ordenados por uma prioridade definida pelo usuário, e uma projeção. Os temas são desenhados com seus respectivos estilos e as coordenadas das camadas apontados pelos temas em questão são transformadas para a projeção da vista. Assim, na geração da textura de dados geográficos, estamos na verdade renderizando uma vista em uma textura.

Para o processo de carga dos dados geográficos para a memória, foi utilizada a biblioteca Tdk (*Terralib Development Kit*), uma biblioteca de auxílio ao desenvolvimento de aplicações geográficas utilizando *Terralib*. A estruturação e o processo da carga dos dados geográficos para o desenho é explicado detalhadamente por Metello et al. (Met07).

3.2 Desenho das Primitivas Geográficas

A abordagem da correção perspectiva exige que o desenho das primitivas geométricas na textura seja feito a cada quadro, o que faz com que o algoritmo de visualização final seja de duas passadas. No primeiro passo, as primitivas geométricas que gerarão a textura são desenhados em um *buffer offscreen* muito comum nas placas de vídeo atuais chamado *framebuffer object*, que permite a renderização de objetos diretamente em uma área da placa reservada à textura. É uma opção eficiente atualmente para texturas que necessitam serem modificadas com certa frequência. No segundo passo, a textura armazenada no *framebuffer* é aplicada ao modelo do terreno.

O desenho das primitivas geográficas utilizando a biblioteca gráfica OpenGL tem algumas peculiaridades que precisam ser levadas em consideração. Por motivos de eficiência, é necessário definir uma modelagem para se utilizar a API de *arrays* que o OpenGL (Shr04) fornece. Essa funcionalidade permite que o usuário da biblioteca defina um espaço da memória onde esteja armazenado seus vértices e outro que define a ordem que esses devem ser desenhados, renderizando blocos de geometrias de uma só vez de forma muito mais eficiente do que o método tradicional utilizando chamadas de glBegin - glEnd.

Para o caso das linhas, para se armazenar vários objetos em apenas um bloco, os vértices do conjunto de linhas são armazenados e a cada um

deles é atribuido um índice incremental. O vetor que define os índices é formado por uma sequência de pares que definem os segmentos de cada linha e são desenhados utilizando a diretiva OpenGL GL_LINES. Assim, é possível desenhar linhas representando vários objetos distintos sem que estejam todas interligadas.

Para os polígonos, são necessários dois conjuntos de definições para a renderização, pois o preenchimento e o contorno são desenhados separadamente. Para o contorno, o tratamento é idêntico às linhas mostrado anteriormente, atentando apenas para o detalhe dos contornos sendo linhas fechadas, adicionando-se um segmento entre o último vértice e o primeiro.

Para o preenchimento do polígono, é necessário um pré-processamento para permitir o desenho. Polígonos não convexos, extremamente comuns em dados geográficos reais, não podem ser desenhados pela sua sequência de vértices usando OpenGL de forma direta. Para essa operação de desenho, o polígono é triangularizado, as definições dos triângulos bidimensionais são armazenadas. O desenho desses triângulos em conjunto se torna a representação visual do polígono armazenado. Os vértices dos triângulos calculados pela função de triangularização (*gluTesselation*) são armazenados em sequência para o desenho, sem a necessidade de um vetor com os índices.

3.3 Transformação e Aplicação da Textura

A transformação aplicada à textura possui algumas restrições para que ela seja válida e aplicável ao terreno de forma correta.

Para que os dados geográficos sejam transformados em uma perspectiva tridimensional, é preciso uma matriz para transformar dados originalmente 2D para coordenadas 3D adequadas. Como normalmente na visualização de terrenos o eixo y representa a altura, desejamos que o dado geográfico existente (x, y) seja convertido para a coordenada tridimensional (x, h, y) onde h é a altura do plano base do terreno, ou seja, a altura do ponto mais baixo de todo o modelo. Para tal, utilizamos a seguinte matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & h \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ h \\ y \\ 1 \end{pmatrix}$$

Um outro detalhe é que os dados geográficos e o modelo de terreno devem representar a mesma região, ou seja, o eixo xz da malha do terreno e dos dados vetoriais devem estar devidamente alinhados.

Para a geração automática de coordenadas de textura, é utilizada a técnica de textura projetiva (Seg92). A transformação deve ser uma matriz que mapeie os valores visíveis para os valores contidos no intervalo [-1, -1, -1] até [1, 1, 1] (cubo canônico), assim como as matrizes que definem a câmera. Assim, dado uma matriz M_{transf}, podemos gerar os seguintes planos para a geração de coordenadas de textura:

$$\begin{pmatrix} 1/2 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot M_{transf} = \begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \\ t_{13} & t_{14} & t_{15} & t_{16} \end{pmatrix}$$

$$SPlane = \begin{bmatrix} t_1 & t_2 & t_3 & t_4 \end{bmatrix}$$

$$TPlane = \begin{bmatrix} t_5 & t_6 & t_7 & t_8 \end{bmatrix}$$

$$RPlane = \begin{bmatrix} t_9 & t_{10} & t_{11} & t_{12} \end{bmatrix}$$

$$QPlane = \begin{bmatrix} t_{13} & t_{14} & t_{15} & t_{16} \end{bmatrix}$$

A primeira matriz é uma combinação de uma translação e uma escala de valor 1/2 em todas as direções. Isso faz com que o espaço do cubo canônico tenha cada uma de suas componentes mapeadas para o intervalo [0, 1], valores válidos para a coordenada de textura. Os planos de geração definidos garantem o mapeamento adequado da textura sobre o terreno. Um ponto positivo dessa aritmética é que várias transformações podem ser testadas.

3.4 Matrizes de Transformação

Esta Seção explica como são calculadas as matrizes para as duas transformações usadas sobre a textura. A primeira, sem correção perspectiva, é útil como parâmetro de comparação. A segunda é a transformação com a correção perspectiva utilizando o conceito do *Light Space* (Wim04).

3.4.1 Matriz sem Correção Perspectiva

Essa transformação é relativamente básica e não pretende corrigir os erros causados pela perspectiva. Ela serve de parâmetro de comparação e para permitir avaliar a melhoria da qualidade devido à correção perspectiva.

A principal dificuldade dessa transformação é definir qual a área da textura necessita ser desenhada para determinada vista do terreno.

O cálculo exato da textura mínima necessária não é algo trivial para ser calculado durante o desenho de cada quadro, portanto foi utilizada uma aproximação. No entanto, essa aproximação não pode ser muito flexível, por dois motivos. Caso a textura seja menor que o necessário, partes do terreno serão apresentadas sem textura. No outro caso, se a textura gerada for muito maior do que a necessária, parte da textura será aplicada em partes não visíveis do terreno, gerando uma textura que não será bem aproveitada, deixando a parte visível com qualidade inferior na texturização.

O algoritmo utilizado é interceptar as retas definidas pelo volume de visão contra o terreno. Como interceptar essas retas contra o terreno de forma exata pode ser uma operação complicada e ainda assim não garantir que todo o terreno visualizado vai ser coberto adequadamente pela textura de dados geográficos, foi utilizada a heurística de interceptar as retas contra a caixa envolvente do terreno. Mas esse procedimento pode causar uma diferença inaceitável para o resultado ideal, como podemos observar na Figura 3.1.

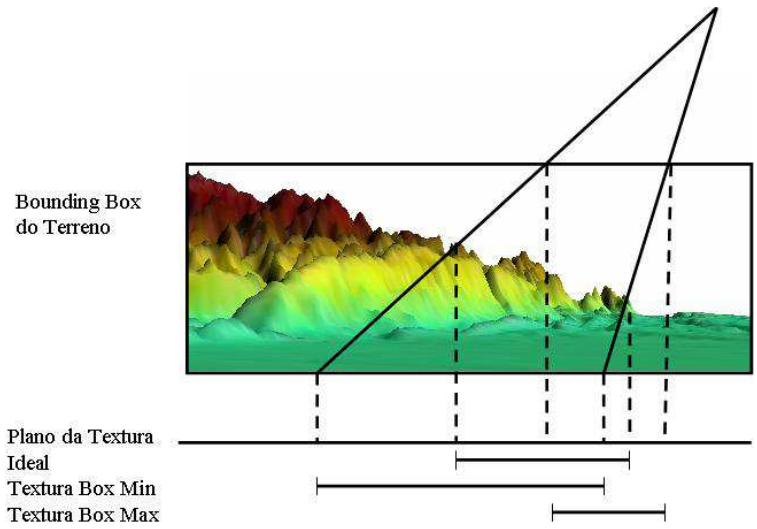

Figura 3.1: Cálculo dos valores para a matriz sem correção perspectiva

Pode-se reduzir a distância do tamanho da textura obtida para o tamanho ideal utilizando uma informação mais local, quando o método de multi-resolução disponibiliza o dado sobre a caixa envolvente de áreas menores do terreno. Obviamente, a caixa envolvente de uma fração menor do terreno é mais precisa que a do terreno como um todo.

Em posse dos pontos onde os raios do volume de visão interceptam as caixas envolventes escolhidas, montamos a área a ser desenhada na textura calculando os pontos extremos minímos e máximos para as coordenadas x e z. Por motivos de padronização, é utilizada uma projeção perspectiva ao invés

da ortográfica natural para o caso, mas gerando uma mesma imagem final. As informações necessárias para montar uma matriz de perspectiva são: o ângulo de abertura na vertical, a razão entre a altura e a largura da imagem, e o valores para os planos de corte *near* e *far*. Os valores escolhidos são 45 graus para o ângulo, 1 para a razão (as texturas são sempre quadradas) e o *near* e o *far* são decididos depois de ser calculado a distância d do observador necessária para visualizar adequadamente a extensão calculada. Os valores x e z do observador são a média dos valores extremos correspondentes obtidos das interseções dos raios do frustum. O valor de d é metade da altura da área de textura calculada vezes a tangente de $45^o/2$. A Figura 3.2 explica mais detalhadamente esses valores.

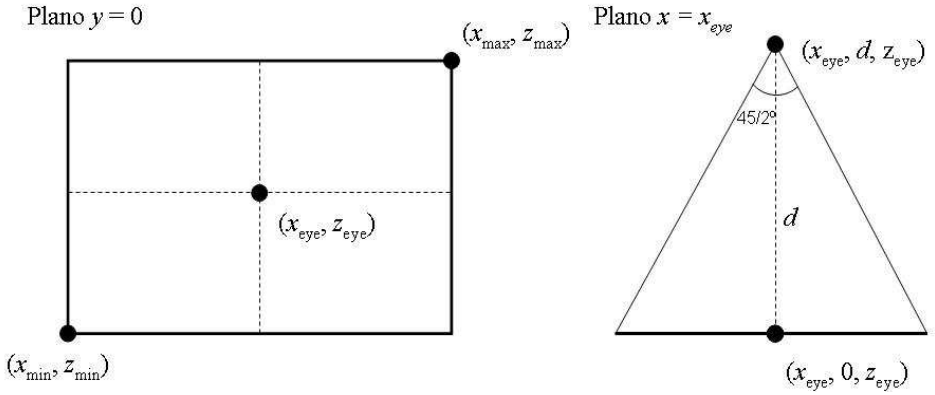

Figura 3.2: Heurística da Bounding Box para o cálculo da área da textura

Considerando as especificações do OpenGL para as funções mostradas a seguir (Shr04), a matriz de transformação utilizada é:

$$M_{transf} = gluPerspective(45, 1, d-1, d+1).gluLookAt(x_{center}, d, z_{center}, x_{center}, 0, z_{center}, 0, 0, 1)$$

3.4.2 Matriz com correção Perspectiva do Light Space Shadow Maps

O processo de obtenção das matrizes do *Light Space* segue a descrição do artigo (Wim04). Primeiramente, é calculado o poliedro B que inclui os raios de "luz" relevantes para uma dada cena, como descrito no artigo. Consideramos a luz hipotética como direcional e vertical em relação ao terreno, ou seja, a direção da projeção da textura.

Seja $\vec{l_{dir}}$ a direção da luz, $\vec{v_{dir}}$ a direção de visualização do observador e O a posição do observador. A partir desses definimos os seguintes vetores:

$$target = O + \vec{l_{dir}}$$

$$\vec{up} = (\vec{l_{dir}} \times \vec{v_{dir}}) \times \vec{l_{dir}}$$

A matriz *lightview* é definida por uma chamada de *gluLookAt* tendo O como posição do olho do observador, *target* como ponto de referência visualizado, e $\vec{up}$ como vetor que aponta para a direção acima do observador. Os valores da perspectiva (n e f) do espaço da luz são calculados da forma que os autores consideraram valores que trazem resultados visuais satisfatórios. Transformamos os pontos do poliedro B pela matriz *lightview*, e obtemos os pontos no espaço da luz. Seja B_{min} e B_{max} os pontos extremos do poliedro pós-transformação. A definição dos valores de n e f segue os passos a seguir:

$$sin_\gamma = \sqrt{1 - (\vec{l_{dir}}.\vec{v_{dir}})^2}$$

$$z_n = \frac{1}{sin_\gamma}$$

$$d = y_{B_{max}} - y_{B_{min}}$$

$$z_f = z_n + d \times sin_\gamma$$

$$n = \frac{z_n + \sqrt{z_n \times z_f}}{sin_\gamma}$$

$$f = n + d$$

Com os valores de n e f especificados, a matriz chamada *lisPSM* é assim definida:

$$lisPSM = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{f+n}{f-n} & 0 & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

No próximo passo, o poliedro B original é transformado pela combinação das matrizes *lisPSM* e *lightView*, gerando o poliedro *B'*. Os pontos extremos do poliedro *B'* são chamados *min* e *max*. Uma matriz é necessária para que o frustum gerado pela transformação para o espaço da luz seja justo à cena. A matriz *fitScene* é responsável por esse ajuste, fazendo que o poliedro *B'* seja transformado para estar ajustado ao cubo canônico, sendo composta por um conjunto de translações e escalas. Primeiramente, se translada o centro da caixa envolvente do poliedro *B'* para o ponto (0, 0, 0). Depois, escala-se cada um dos eixos de forma que em cada um das direções a caixa resultante fique com tamanho 2, formando assim uma caixa envolvente com limites no cubo canônico. Assim, a matriz *fitScene* é composta pelas duas matrizes a seguir:

$$fitScene = \begin{pmatrix} \frac{2}{x_{max}-x_{min}} & 0 & 0 & 0 \\ 0 & \frac{2}{y_{max}-y_{min}} & 0 & 0 \\ 0 & 0 & \frac{2}{z_{max}-z_{min}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & \frac{-(x_{max}+x_{min})}{2} \\ 0 & 1 & 0 & \frac{-(y_{max}+y_{min})}{2} \\ 0 & 0 & 1 & \frac{-(z_{max}+z_{min})}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

O resultado final da matriz de transformação é:

$$M_{transf} = fitScene.lisPSM.lightview$$