Visualização de Grandes *Geobodies* em Tempo Interativo com suporte a Filtragem por Componentes e Valores

Leonardo Novaes, Carlos André Campos, Pedro Mário Silva, Waldemar Celes
Tecgraf/PUC-Rio – Departamento de Informática
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, Brazil
email: {leomaia, ccampos, pmario, celes}@tecgraf.puc-rio.br

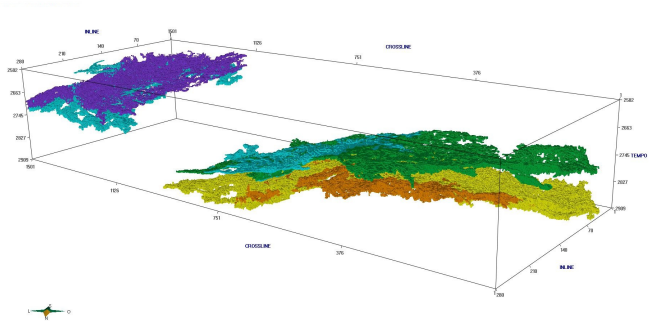


Figure 1. Geobody de múltiplas classes.

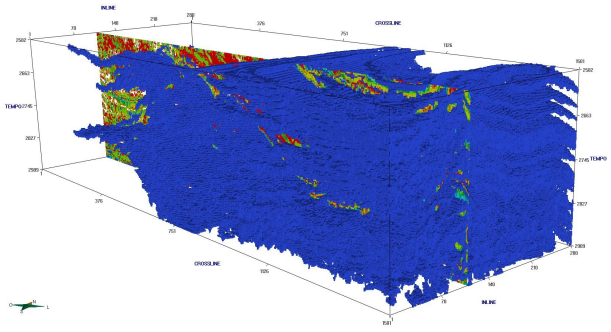


Figure 2. Geobody de uma grande componente conexa.

***Abstract*—Este artigo propõe um método para renderização interativa de dados sísmicos classificados e agrupados por componentes conexas. O método proposto provê suporte às filtragens necessárias para a interpretação geológica. Propomos uma redução do volume de dados transferidos para a GPU codificando cada bloco de 2x2x2 de 8 voxels em uma palavra de 8-bits, emitindo-os como pontos para o pipeline gráfico, reduzindo assim a carga de geometria da aplicação. No programa de fragmentos, os cubos são renderizados via um eficiente algoritmo de traçado de raios.**

***Keywords*-visualização de dados sísmicos; visualização volumétrica discreta; traçado de raio;**

***Abstract*—This paper proposes a method for interactive rendering of classified seismic data grouped by connected components. The proposed method provides filtering needed for better geological interpretation. We propose a reduction of the data volume transferred to the GPU by coding each 2x2x2 block of 8 voxels in one 8-bits word, emitting them as points to the graphics pipeline thereby reducing the geometric load of the aplication. At the fragment program, the cubes are rendered through an efficient ray-tracing algorithm.**

***Keywords*-seismic data visualization; discrete volume rendering; ray-tracing;**

I. Introdução

A indústria de petróleo define uma grande fatia do mercado mundial. Para aumentar a precisão da exploração é possível extrair dados sísmicos da área de interesse para futuras interpretações. A visualização tridimensional destes dados pode ser utilizada como uma ferramenta de interpretação geológica [1], [2], [3], [4].

A interpretação sísmica estrutural trata basicamente da identificação e mapeamento de horizontes (interface entre camadas) e falhas geológicas, que são variedades de dimensão 2 (superfícies). Os *geobodies* (ou corpos geológicos), por sua vez, são regiões tridimensionais do espaço que estão relacionadas com outras feições geológicas, como por exemplo canais, leques, deltas, turbiditos, etc. Os *geobodies* têm um papel muito importante na interpretação sísmica devido a habilidade de modelar a geometria destes corpos geológicos.

A classificação de *geobodies*, em geral, é feita de forma automática. O processo de classificação dos voxels pode ser feita por uma abordagem de rede neural, geoestatística, etc. A classificação é feita em cima de um ou mais volumes de atributos sísmicos. O resultado da classificação é um volume onde cada voxel tem o valor inteiro da classe a que ele pertence.

O resultado bruto do processo de classificação é complexo e pode chegar a ter bilhões de voxels, pois tem as mesmas dimensões do dado de entrada. A identificação de componentes conexas das classes e a filtragem por tamanho de cada componente conexa são ferramentas essenciais para que o intérprete consiga extrair alguma informação geológica relevante do resultado da classificação. A determinação da conectividade é importante para esclarecer, por exemplo, se o reservatório de petróleo está segmentado ou não, o que

influencia na estratégia de posicionamento de poços para exploração e injeção de fluidos. A filtragem por tamanho da componente conexa permite a eliminação de componentes muito pequenas que estão relacionadas com ruído contidos nos dados de entrada, além disso permite eliminar componentes muito grandes (que podem se estender por todo o volume), que na verdade são artefatos gerados pelo próprio processo de classificação.

Contribuição: Apresentamos neste trabalho um método para renderizar de maneira interativa de objetos do tipo *geobody* como um aglomerado de cubos discretos, contemplando filtragem por classe e por tamanho de componentes conexas, proporcionando uma melhor interpretação geológica tridimensional. O primeiro problema abordado é a redução do montante de dados transferidos para a VRAM. Isto é feito codificando cada bloco de 2x2x2 do volume original em metacubos, usando palavras de 8-bits de acordo com a disposição espacial dos cubos ali contidos.

A expansão de cada metacubo em 1 a 8 cubos é feita por programação em placa gráfica. São gerados 256 programas de geometria (*geometry shaders*) codificados para cada configuração particular. O esforço geométrico é balanceado pela utilização de pontos de diferentes tamanhos como primitivas de desenho. A simplicidade e a eficiênca do uso de pontos como primitivas gráficas já foram explorados em [5] e aprimorados em [6], [7], entre outros.

O programa de geometria cria pontos associados aos cubos discretos que devem ser renderizados, via traçado de raios no programa de fragmentos. Assume-se iluminação com modelo de luz direcional difusa, o que possibilita o cálculo de cor em préprocessamento na CPU.

II. Trabalhos Relacionados

Técnicas de visualização de dados sísmicos tridimensionais tem sido amplamente pesquisadas. Tavares et al. [8] utilizaram técnicas de visualização volumétrica para seleção de componentes pertencentes ao *geobody*, utilizando uma variação [9] do algorítimo clássico de *marching cubes* [10]. Silva [11] et al. propuseram o uso de funções de transferência bidimensionais para renderizar estes dados. Os dados sísmicos atuais são muito maiores, demandando constantes avanços nas técnicas de visualização.

Muitos métodos clássicos abordaram a visualização de voxels através de *cuberille* (cubos opacos). Uma revisão destes métodos foi apresentada por Roberts [12]. Uma demanda grande para redução do esforço geométrico destas visualizações clássicas existe desde suas primeira aparições em dados médicos [13]. Contudo, a complexidade dos modelos cresceu muito com a evolução da tecnologia, encorajando pesquisadores a procurarem outras primitivas de desenho alternativas a triângulos.

O uso de pontos como primitiva gráfica foi introduzido por Levoy [5]. Técnicas baseadas em ponto [14] tentam desacoplar a complexidade da cena, adotando estratégias no espaço da imagem. Com o advento da programação em placa gráfica, isto pode ser estendido e otimizado [7].

Toledo e Levy [15] apresentaram uma maneira eficiente de renderizar objetos usando pontos como primitivas e traçados de raio por fragmento. Uma geometria simples que engloba a imagem do objeto desejado é utilizada como base do traçado de raios, sendo também aplicado a renderização de modelos de instalações industriais [16]. Na mesma linha, Sigg et al. [17] propuseram renderizar quádricas a partir da renderização de pontos e traçado de raios na GPU.

Diferentes estratégias [12] de visualização volumétrica também foram exploradas. Para conseguir aplica-las a dados das dimensões do *geobody*, um método baseado em cache e paginação inteligente foi utilizado em [18]. Um método com uma abordagem parecida foi aplicado na engine GigaVoxels [19], que estrutura seus dados de forma a otimizar o uso da memória usando um sistema de nível de detalhamento e anti-aliasing enquanto reduz o esforço computacional da GPU.

Plate et al. [18] definem taxas de visualização interativas como uma questão muito importante na interpretação de dados sísmicos. Durante o movimento da câmera, [18] e [19] usam uma estrutura de nível de detalhamento para minimizar a transferência de dados para a GPU, diminuindo a qualidade de imagem. Propomos uma solução parecida onde uma versão com menos detalhe do *geobody* é desenhada durante o modo "rascunho" para garantir essas taxas de visualização interativa na manipulação do modelo.

III. Método proposto

O objetivo do método proposto é visualizar de forma eficiente *geobodies* (corpos geológicos) definidos por cubos discretos classificados a partir de um volume sísmico. Nossa proposta visa reduzir a quantidade de dado transferido para a VRAM e minimizar a carga de geometria destes modelos massivos.

A. Compressão

O *geobody* é um conjunto de voxels, com valores de classificação associados, dispostos espacialmente numa grade tridimensional. Esta grade é chamada de espaço *cuberille* [20], definido por voxels igualmente espaçados entre planos ortogonais. Desta forma, cada voxel existente preenche um cubo discreto na grade. Voxels vizinhos conectados estão agrupados em componentes conexas.

Uma das principais metas desta proposta é reduzir a quantidade de dados transferidos para a VRAM. O espaço *cuberille* utilizado para representar o dado pode ser compactado agrupando cada bloco de 2x2x2 elementos, codificando a distribuição espacial dos voxels ali contidos. Para cada voxel deste bloco podemos atribuir 1 se este existe ou 0 caso contrário. A partir desta codificação, cada bloco de 8 voxels é representado por uma palavra de 8 bits, gerando $2^8 = 256$ possíveis configurações, como exemplificado na Figura 3.

Cada bloco de 8 voxels pode agora ser representado como um único metavoxel que precisa ser posteriormente expandido no número correto de voxels (de 1 a 8). Durante uma etapa de pré-processamento, cada grupo de 8 voxels vizinhos são categorizados pela distribuição espacial em uma das 256

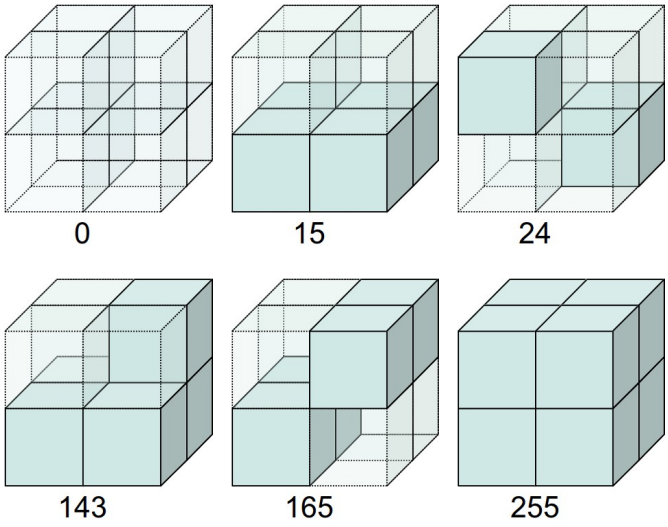


Figura 3. Exemplos de configurações de metacubos.

configurações. Salientamos que o caso 0, onde nenhum cubo deve ser desenhado, não é armazenado nem transferido para o pipeline gráfico.

B. Estrutura de Dados

Um conceito importante para a interpretação do volume sísmico é a definição de componentes conexas. Voxels de uma mesma classe que estão dispostos espacialmente de acordo com uma regra de conectividade são agrupados em uma mesma componente durante o rastreamento. Como introduzido anteriormente o volume sísmico de onde o *geobody* é extraído é de natureza ruídosa. A presença destes ruídos resultam na formação de componentes conexas pequenas. O valor geológico do objeto provém da análise de componentes conexas tipicamente grandes. Com o objetivo de separar a parte que interessa do *geobody*, um filtro por tamanho de componente conexa é apresentado, como ilustra a Figura 5 . Como cada classe possui relevância distinta das demais, nossa estrutura de dados contempla também um filtro por valor de classificação.

Para assegurar que a aplicação de filtros na visualização de *geobodies* seja feita de forma eficiente, propomos uma estruturação de *vertex arrays*. Cada valor de classificação possui 256 vetores, armazenando os metavoxels correspondentes a cada configuração. Tipicamente, a interpretação é focada em um número reduzido de classes de cada vez, uma vez que o objetivo é justamente observar uma região menor do dado que representa o reservatório, canal ou outra região de interesse.

O filtro por tamanho de componentes conexas é feito através do agrupamento dos metavoxels de cada componente nos 256 vetores ordenados por tamanho de componente como ilustrado em Figura 4 . No momento do desenho escolhe-se um pedaço do vetor começando no primeiro voxel da menor componente visível e terminando no último voxel da maior componente visível. Para encontrar os índices que representam o intervalo

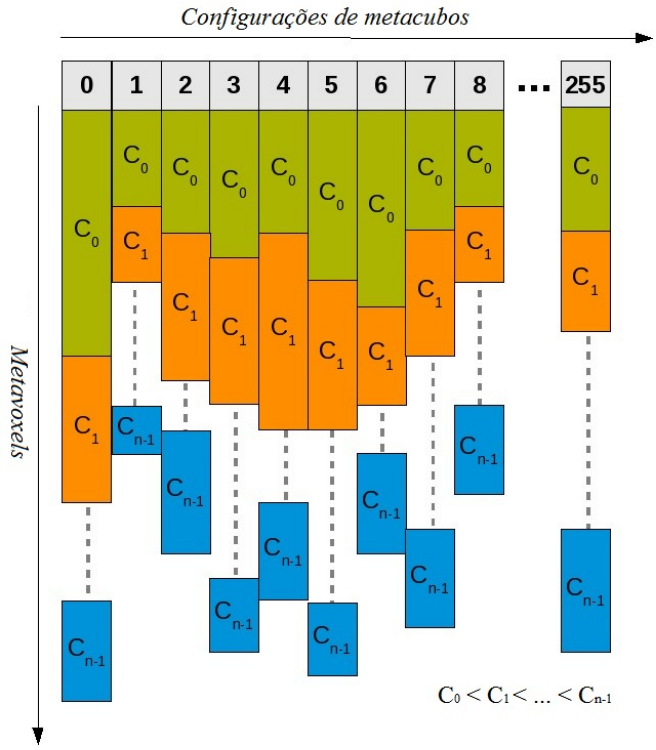


Figura 4. Organização dos vertex arrays para cada classificação. Cada configuração representa um vetor ordenado por tamanho de componente.

desejado, utiliza-se uma busca binária em um vetor auxiliar também ordenado contendo índices de cada componente.

C. Eliminação de Metavoxels Oclusos

Como o principal objetivo deste método é a redução do fluxo de dados entre RAM e VRAM, é fácil perceber que componentes conexas muito grandes possuem muitos metavoxels que não serão visíveis, pois estão oclusos por seus vizinhos. Apresentamos uma estratégia para identificar e retirar voxels oclusos dos vetores transferidos para a placa de vídeo. O principal problema aparece após a filtragem: voxels oclusos podem ficar expostos após seus vizinhos terem sido retirados da visualizacão. Recalcular as oclusões após a filtragem é inviável pelo tamanho do dado. A filtragem deve ocorrer rapidamente sem pausas para processamento de forma que não atrapalhe a visualização.

Um voxel na fronteira é indentificado pela ausência de um de seus vizinhos diretos. De forma análoga, todo voxel que possui seus vizinhos é candidato a ser ocluso. Como componentes conexas são definidas por voxels adjacentes, casos de configuração de metavoxel não completos podem não pertencer a mesma componente que seus vizinhos. Desta forma apenas o caso 255, onde todos os subcubos existem, são candidatos a estarem oclusos, mesmo após a execução de filtragens.

Cada metavoxel do caso 255 candidato a oclusão precisa ter sua vizinhança analisada. Esta análise é feita usando uma

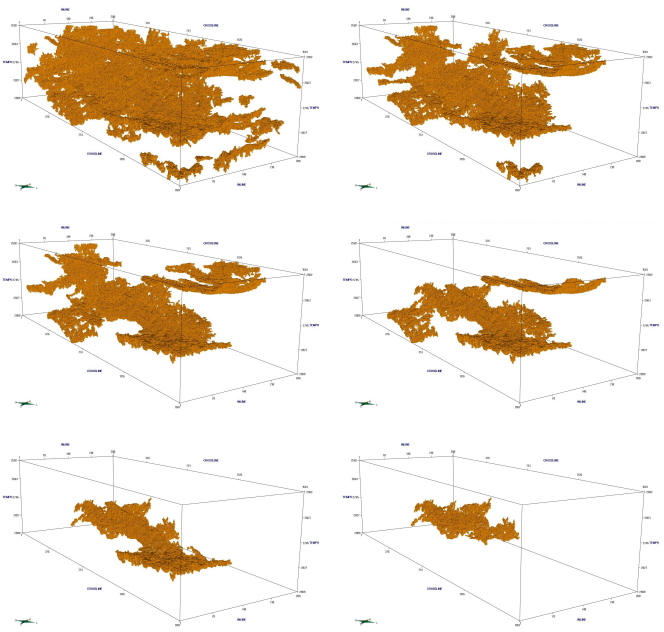


Figura 5. Exemplo da filtragem por tamanho de componente conexa.

máscara de bits aplicada a cada metavoxel vizinho, identificando se os voxels vizinhos diretos existem. A Figura 6 mostra uma redução em 2D do problema, ilustrando as operações de bits feitas para identificar a oclusão dos casos 255.

D. Vertex e Geometry Shaders

Dados os *vertex arrays* por configuração, propomos uma programação em placa gráfica capaz de converter os metavoxels em cubos, de 1 a 8, eficientemente. Nossa abordagem consiste em usar a primitiva de pontos para redução do esforço geométrico. Cada metavoxel é enviado para o pipeline gráfico através da primitiva ponto.

O programa de vértice (*vertex shader*) é encarregado de computar um tamanho de ponto suficiente para cobrir, no espaço da imagem, inteiramente todos os cubos que serão desenhados para a configuração em questão. Aproveitando o fato de que operações com matrizes são executadas com bastante eficiência, projeta-se um cubo que representa o caso 255 na tela e computa-se a largura e altura em pixels deste objeto. O tamanho do ponto que engloba completamente a projeção de qualquer cubo, gerado por qualquer caso, é definido pelo maior valor entre estas dimensões em pixels dividida por dois.

O papel do programa de geometria (*geometry shader*) é emitir de 1 a 8 pontos de acordo com cada configuração de metavoxel. Programas de geometria são conhecidos por não serem muito eficientes ao tratar muitos parâmetros. *Shaders* que dependem de muitos parâmetros possuem muitos caminhos diferentes que podem ser tomados a cada execução do programa, gerando um problema de paralelismo. Testes mostram que gerar 256 programas, um para cada configuração, chaveando entre eles durante o desenho, foi mais eficiente que manter

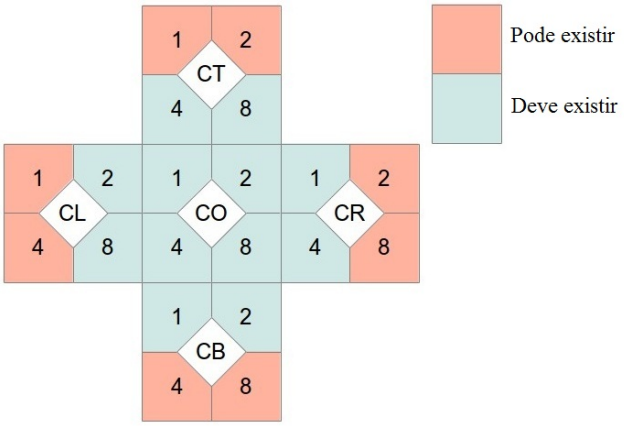


Figura 6. Redução 2D do problema da identificação de metacubos oclusos.

um número restrito de *shaders* que tratam muitos parâmetros. Uma redução de 256 para 15 casos utilizando simetrias como na técnica de *marching cubes* não foi constatada durante os testes.

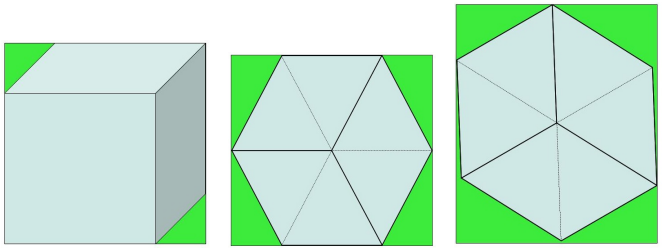
E. Programa de Fragmento

Outros trabalhos [15], [16], [17] obtiveram eficiência ao traçar raios no programa de fragmento, renderizando primitivas implícitas através de geometrias mais simples cuja projeção engloba a primitiva desejada. Utilizamos a mesma abordagem para converter pontos em cubos, um algoritimo simples de interseção de raio com cubo alinhado ao eixo é executado para cada fragmento do ponto, calculando se aquele pixel pertence ou não ao subcubo sendo avaliado. A área rasterizada onde os raios não interceptam o cubo representa uma pequena porção da área total do ponto como ilustrado em Figura 7.

No sistema de interpretação de dados sísmicos, onde esta abordagem foi implementada, o modelo é, em geral, iluminado com luz direcional difusa. O intérprete pode manipular a direção da luz para visualizar profundidade e características volumétricas. Como todo cubo renderizado é alinhado ao eixo, a cor de cada face é a mesma para todos eles. Para reduzir ainda mais o esforço da GPU, todas as cores e efeitos de iluminação são calculados na CPU e passados como parâmetro para o programa de fragmento. Uma vez que um raio traçado interceptou um cubo, computa-se qual face foi interceptada primeiro e define-se a cor do pixel a partir dos valores de cor computados na CPU.

F. Modo "Rascunho"

O principal objetivo do visualizador interativo é obter maior eficiência enquanto o usuário está manipulando o modelo. A interpretação do *geobody* consiste em primeiramente detectar e filtrar grandes componentes conexas de determinada classificação e, depois, alterar a posição da câmera buscando um ponto de vista adequado para a interpretação da estrutura tridimensional. Uma vez definido um ponto de vista, outras ferramentas de interpretação são utilizadas (ex. picking). Para assegurar a visualização interativa criamos um modo



(a) Caso alinhado à tela. (b) Caso com rotação de 45 ° em cada eixo. (c) Caso qualquer.

Figura 7. Casos de pontos englobando cubos em diversas perspectivas. área em verde representa raios traçados que não interceptam o voxel.

"rascunho" de visualização para quando a câmera está em movimento.

Durante a manipulação do modelo, o geólogo precisa apenas saber a forma crua do objeto e ter noção do tamanho das componentes. Analisando nossos casos de metavoxels, podemos intuitivamente determinar que o interior de grandes componentes conexas serão compostos do caso 255, onde todos os 8 cubos existem. Partindo deste princípio, assumimos que desenhar apenas o caso 255 pode ser suficiente para ter-se uma representação aproximada do dado.

Nosso modo "rascunho" é constituído pela renderização de apenas casos 255 durante a movimentação. Quando a câmera se mantém parada e uma interpretação estática ocorre, nós renderizamos o *geobody* de maneira normal, com todos os casos de metavoxels.

IV. Resultados e Discussões

Validamos o desempenho do nosso método renderizando objetos *geobody* extraídos de dados sísmicos reais previamente classificados. Cada valor de classificação dos corpos rastreados foram renderizados separadamente para avaliar o funcionamento do método para diferentes tipos e tamanhos de componentes conexas. Registramos o número de componentes e voxels para cada classe e então computamos a taxa de quadro por segundo (FPS) de ambos modos de visualização: modo normal e modo "rascunho". Para avaliar o ganho em redução de memória, comparamos o número de metavoxels com o número de cubos efetivamente renderizados.

Para execução de nossos experimentos, usamos uma estação Z800 equipada com processador Xeon x5690 3.47Ghz (12 núcelos), 70 GB de RAM e placa gráfica nVidia Quadro 6000 com 6GB de VRAM. Todas as implementações foram feitas usando OpenGL 4.1. Os testes foram feitos gerando imagens na resolução de 1870x1050, partindo do princípio de que a maioria das interpretações sísmicas são feitas usando resoluções altas.

A. Desempenho

Dois dados sísmicos de diferentes tamanhos foram utilizados para execução do teste de desempenho. O primeiro dado possui dimensões de 2496x1828x954, representando um volume de classificação sísmica de 4.05GB. Como é um volume relativamente pequeno, mensuramos todas as nove classificações de valores nos modos normal e "rascunho". Como a Tabela I demonstra, obtivemos sucesso em visualizar um máximo de 123.5 milhões de cubos a uma velocidade interativa de 5.45 FPS no modo de visualização normal e 11.63 FPS no modo "rascunho".

Tabela I
Tempos de renderização: dado sísmico de 2496x1828x954.

Classe	nComponentes	nVoxels	FPS	Draft FPS
1	157.037	107.021.791	5.99	13.11
2	214.677	28.427.516	17.49	119.09
3	194.220	123.457.001	5.45	11.63
4	266.779	33.921.945	13.36	125.66
5	466.430	5.568.030	53.62	125.76
6	284.468	44.096.097	10.35	124.74
7	104.883	52.463.815	11.45	31.52
8	349.501	5.449.251	42.24	125.63
9	111.767	73.873.970	8.32	22.87

Tabela II
Tempos de renderização: dado sísmico de 8247x14317x1751.

Classe	nComponentes	nVoxels	FPS	Draft FPS
7	3.572.795	266.993.894	1.79	23.50
8	7.374.567	175.130.235	2.07	24.5

O segundo dado de teste é de dimensão 8247x14317x1751, resultando em 192.54GB de volume de 8 bits de classificação. Considerando o tamanho deste dado, apenas dois valores de classificação foram rastreados. Nossos resultados apresentados na Tabela II foram satisfatórios, renderizando quase 267 milhões de cubos a uma taxa interativa de 1.79 FPS. Durante o modo "rascunho", esta velocidade aumentou para 23.50 FPS. Baseando-se nestes resultados, conseguimos validar que nosso método pode realmente renderizar objetos *geobody* de grande porte a taxas interativas, assegurando aumento de desempenho durante a manipulação do modelo (modo "rascunho").

B. Uso da Memória

No primeiro dado de teste, nove classes estavam presentes no *geobody*. Uma análise da redução de memória pode ser feita gerando um histograma de casos de metavoxel existentes em cada classe. Para simplificar o resultado apresentado na Tabela III, escolhemos apenas algumas classes representativas. A penúltima linha da tabela mostra o número de metavoxels oclusos identificados. Note que é um número significativo, validando nossa estratégia de detecção de oclusão.

A taxa compressão, última linha da tabela, é determinada pelo número total de metavoxels dividido pelo total de voxels gerados. Os resultados mostram que, na maioria dos casos, mais de 80% do tamanho original foi comprimido. Em classes que eram esparças demais, com um número alto de pequenas componentes conexas, a redução alcançou um valor de 48.02%. Como classes esparças possuem pouca relevância geológica, eles são em geral filtrados e não visualizados durante a interpretação sísmica padrão. Nossa melhor taxa de redução ocorre em casos com grandes componentes conexas, com grande importância geológica. As classes mais relevantes estão presentes na Tabela III .

Tabela III
HISTOGRAMA DE CASOS DE METAVOXELS MOSTRANDO REDUÇÃO DE DADOS TRANSFERIDOS PARA A GPU – VALORES REPRESENTAM O NÚMERO DE METAVOXELS.

nCubos	Classe 2	Classe 3	Classe 6	Classe 8	Classe 9
1	852.178	599.884	384.297	3.245	9.184
2	1.034.178	673.381	310.567	1.797	7.039
3	639.286	425.086	219.674	825	4.241
4	1.312.076	784.421	252.749	496	4.856
5	604.199	399.600	171.854	135	2.932
6	970.363	592.040	208.272	76	3.526
7	712.629	503.743	197.660	20	2.661
8	3.017.039	3.344.878	344.395	18	3.486
Oclusos	1.991.769	2.516.569	266.043	4	2.077
Compressão	85.12%	88.60%	79.56%	48.02%	73.97%

V. CONCLUSÃO

Apresentamos um método eficiente para visualização de objetos sísmicos do tipo *geobody*. Modelos de *geobodies* devem ser visualizados por algoritmos com suporte a filtragem por classificação e por tamanho de componente conexa. O método proposto atende a estes requisitos. Como trata-se de um modelo representado por cubos discretos não interpoláveis, técnicas convencionais de visualização volumétrica não se aplicam. Conforme apresentado, o método proposto apresenta três características importantes: redução do volume de dados, emissão de pontos como primitiva e traçado de raios para renderização de cubos discretos.

Analisando os resultados apresentados, concluímos que nossa codificação dos voxels em configurações de metavoxels obteve sucesso ao reduzir o montante de dado transferido para a VRAM por um expressivo fator máximo de 88%. Esta redução de transferência de dado refletiu no desempenho alcançado: taxas interativas para modelos gigantes.

Os resultados dos experimentos comprovam a eficiência do método proposto para visualização de *geobodies*, com devido suporte a filtragem por classificação e por tamanho de compenente conexa. O modo "rascunho" proposto permite aumentar de forma significativa a taxa de renderização, possibilitando fácil manipulação do modelo. Antes deste algorítmo era inviável a visualização de geobodies grandes.

REFERÊNCIAS

[1] G. D. Kidd, "Fundamentals of 3-d seismic volume visualization," in *The Leading Edge*, 1999, pp. 702–709.

[2] G. A. Dorn, "Modern 3-d seismic interpretation," in *The Leading Edge*, 1998, vol. 17, pp. 1262–1272.

[3] D. Gao, "3d seismic volume visualization and interpretation: An integrated workflow with case studies," in *Geophysics*, 2009, vol. 74, pp. w1–w12.

[4] A. R. Brown, *Interpretation of Three-Dimensional Seismic Data*, 1999, vol. 42.

[5] M. Levoy and T. Whitted, "Abstract the use of points as a display primitive," 1985.

[6] S. Rusinkiewicz and M. Levoy, "Qsplat: A multiresolution point rendering system for large meshes," 2000.

[7] L. Ren, H. Pfister, and M. Zwicker, "Object space EWA surface splatting: A hardware accelerated approach to high quality point rendering," in *Computer Graphics Forum (Eurographics 2002)*, vol. 21, no. 3, Sep. 2002, pp. 461–470.

[8] G. Tavares, R. Santos, H. Lopes, T. Lewiner, and A. W. Vieira, "Topological reconstruction of oil reservoirs from seismic surfaces," in *International Association for Mathematical Geology*, september 2003.

[9] T. Lewiner, H. Lopes, A. W. Vieira, and G. Tavares, "Efficient implementation of marching cubes cases with topological guarantees," *Journal of Graphics Tools*, vol. 8, no. 2, pp. 1–15, december 2003.

[10] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *COMPUTER GRAPHICS*, vol. 21, no. 4, pp. 163–169, 1987.

[11] P. Silva, M. Machado, and M. Gattass, "3d seismic volume rendering," 2003.

[12] J. C. Roberts, "An overview of rendering from volume data - including surface and volume rendering," Tech. Rep., 1993.

[13] C. Montani, R. Scateni, and R. Scopigno, "Discretized marching cubes," in *Visualization '94 Proceedings*. IEEE Computer Society Press, 1994, pp. 281–287.

[14] J. P. Grossman and W. J. Dally, "Point sample rendering," in *In Rendering Techniques '98*. Springer, 1998, pp. 181–192.

[15] R. Toledo and B. Lévy, "Extending the graphic pipeline with new gpu-accelerated primitives," Tech. Rep., 2004.

[16] R. Toledo and B. Levy, "Visualization of industrial structures with implicit gpu primitives," in *Proceedings of the 4th International Symposium on Advances in Visual Computing*, ser. ISVC '08, 2008, pp. 139–150.

[17] C. Sigg, T. Weyrich, M. Botsch, and M. H. Gross, "Gpu-based ray-casting of quadratic surfaces." in *SPBG*, M. Botsch, B. Chen, M. Pauly, and M. Zwicker, Eds. Eurographics Association, 2006, pp. 59–65.

[18] J. Plate, M. Tirtasana, R. Carmona, and B. Fröhlich, "Octreemizer: a hierarchical approach for interactive roaming through very large volumes," in *Proceedings of the symposium on Data Visualisation 2002*, ser. VISSYM '02, 2002, pp. 53–ff.

[19] C. Crassin, "Gigavoxels: A voxel-based rendering pipeline for efficient exploration of large and detailed scenes," Ph.D. dissertation, UNIVERSITE DE GRENOBLE, July 2011.

[20] L. shyang Che, G. T.Herman, R. Reynolds, and J. Udupa, "Surface shading in the cuberille environment," *IEEE Computer Graphics and Applications*, vol. 5, 1985.