2
Detalhamento do Processo Adotado

Neste capítulo, será apresentado como foram definidas as técnicas e passos do processo para a utilização específica do *Surface Splatting* com dados reais de horizontes sísmicos 3D interpretados. Após a apresentação da técnica de *Surface Splatting*, partiremos dos dados disponíveis e passo a passo, analisaremos os melhores caminhos para a utilização do método, de forma a respeitar as características dos dados e privilegiar aspectos desejáveis na sua visualização.

2.1
Surface Splatting

A técnica de renderização de superfícies conhecida como *Surface Splatting*, utilizada neste trabalho para a visualização de horizontes, foi introduzida por Zwicker et al., em (Zwicker01), como uma forma de visualização de superfícies opacas ou transparentes, a partir de nuvens de pontos, que representam superfícies.

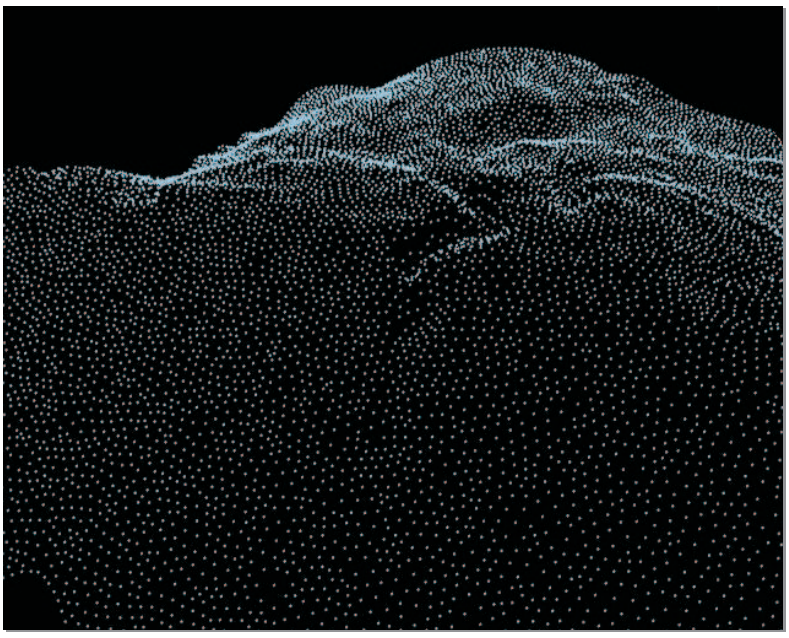

Figura 2.1: Nuvem de pontos de um horizonte sísmico 3D interpretado (fundo do mar).

O objetivo proposto do método é mapear texturas contínuas em superfícies baseadas em pontos, utilizando para isso uma adaptação da teoria de reamostragem de Heckbert para reconstruir a textura no espaço de tela.

Para a aplicação da técnica, é esperado como entrada um conjunto de pontos dispostos de forma não regular no espaço 3D, sem qualquer informação de conectividade associada, com a posição e normal especificadas. Para cada ponto existe a informação necessária à iluminação, como os componentes da cor: vermelho, verde e azul (RGB), associada a uma função de base radial, radialmente simétrica, que representa uma variação contínua e suave desta informação na área de influência especificada para o ponto. Esta função de base radial deve ter suporte local, limitando a influência de cada ponto a uma pequena vizinhança do mesmo. Chamaremos de *splat*, neste trabalho, cada ponto com suas respectivas informações associadas.

Para calcularmos o resultado final das contribuições para um ponto qualquer da superfície, que pertença ou não ao conjunto de *splats*, é feita uma parametrização local, onde o ponto e os *splats* da vizinhança são aproximados para um mesmo plano e o resultado é o somatório das influências de cada *splat* para aquele ponto.

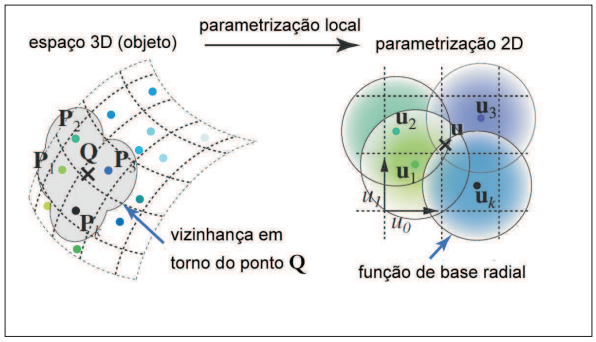

Figura 2.2: Parametrização 2D e área de influência de um conjunto de pontos, adaptado de (Zwicker01)

Em linhas gerais, a função radialmente simétrica é projetada (aproximadamente) no plano de tela, resultando em uma elipse que passa, então, por um filtro de reamostragem, acumulando os resultados gerados no espaço de tela.

O processo de reamostragem pode ser visto como análogo ao processamento de sinais. A função contínua é reconstruída a partir das amostras fornecidas, para depois ser reamostradada no espaço de imagem, de forma atender ao teorema de Nyquist para minimizar a perda de informação no processo. E, assim como no processamento de sinais, um filtro de passa-baixas deve ser utilizado, para evitar a ocorrência de *aliasing* após a reamostragem.

O primeiro passo do processo, então, é a reconstrução da função contínua, baseando-se nas amostras originais. Para isso um filtro de reconstrução deve

ser utilizado.

Podemos definir um filtro como um operador que, a partir de um sinal - no caso uma função - de entrada, gera um sinal modificado como saída. Um filtro linear (onde mudanças de escala no sinal de origem geram mudança de igual natureza na resposta), invariante no tempo (onde deslocamentos no espaço no sinal de origem geram igual deslocamento na resposta), é caracterizado pela resposta de impulso $h(x)$, que corresponde à saída gerada quando o sinal de entrada é um impulso (um sinal que é igual a 1 quando $x = 0$ e 0, caso contrário). A resposta de impulso $h(x)$ é também conhecida como *kernel* do filtro. A resposta a um filtro linear invariante no tempo a qualquer sinal de entrada $f(x)$, é dada pela convolução (aqui representada pela operação $\otimes$) de $f(x)$ e $h(x)$. Então, para um filtro $h(x)$, temos que:

$$b(x) = \int_{-\infty}^{\infty} f(t)h(x-t)dt = (f \otimes h)(x) \tag{2-1}$$

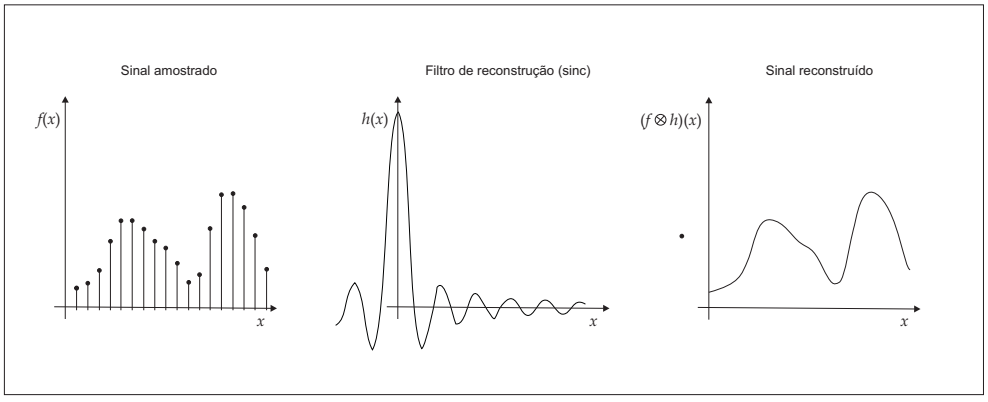

Figura 2.3: Aplicação de filtro de reconstrução

Os filtros escolhidos para serem utilizados no *Surface Splatting* são filtros de Gaussiana 2D, devido à sua distribuição espectral e propriedades analíticas, que possibilitam que mapeamentos lineares e convoluções sejam obtidos de forma eficiente. Uma Gaussiana 2D pode ser definida como:

$$g_{\mathrm{V}}(x) = \frac{1}{2\pi\sqrt{|\mathrm{V}|}} e^{-\frac{1}{2}x^T \mathrm{V}^{-1}x} \tag{2-2}$$

onde x é um vetor coluna 2x1 e V é uma matriz simétrica, 2x2, que representa a *matriz de variância* (e $|\mathrm{V}|$ seu determinante).

O mapeamento linear M de uma Gaussiana com matriz de variância V é uma outra Gaussiana com matriz de variância MVM^T. E a convolução de duas Gaussianas g_{V} e g_{W} com respectivas matrizes de variância V e W é uma outra Gaussiana com matriz de variância igual a soma das duas matrizes:

$$(g_{\mathrm{V}} \otimes g_{\mathrm{W}})(x) = g_{\mathrm{V+W}}(x) \tag{2-3}$$

Introduzidos os conceitos básicos necessários, voltamos para a definição do filtro de reconstrução utilizado no *Surface Splatting*.

O *kernel* 2D do filtro de Gaussiana, utilizado na reconstrução, é definido para cada ponto de entrada, no plano tangente ao ponto. Para definir a parametrização 2D do plano tangente no ponto i, calculamos os vetores tangentes u_i e v_i, a partir da posição e normal do ponto, e definimos o *kernel* de reconstrução como sendo a Gaussiana:

$$g_{\mathrm{R}_i}(\mathrm{t}) = \frac{1}{2\pi\sqrt{|\mathrm{R}_i|}} e^{-\frac{1}{2}\mathrm{t}^T \mathrm{R}_i^{-1} \mathrm{t}} \tag{2-4}$$

onde t são as coordenadas no plano tangente, representadas como um vetor coluna: $\mathrm{t} = (\mathrm{t}_0, \mathrm{t}_1)^T$.

A transformação dos pontos nas coordenadas locais 2D para o espaço de imagem 3D seria feita, em condições normais, através de uma transformação projetiva. Neste caso, entretanto, se aplicássemos a projeção perspectiva ao *kernel* de reconstrução, o resultado deixaria de ser uma Gaussiana, dificultando a posterior aplicação de um pré-filtro.

Para contornar o problema é feita uma aproximação afim para o mapeamento projetivo, de forma a garantir que o *kernel* de reconstrução resultante, no espaço de imagem, ainda seja uma Gaussiana.

O novo *kernel* de reconstrução, no espaço da tela, é então calculado como:

$$r'_i(x - p'_i) = |\mathrm{J}_i| g_{\mathrm{J}\mathrm{V}_i \mathrm{J}^T}(x - p'_i) \tag{2-5}$$

onde p'_i é o ponto no espaço de imagem, $\mathrm{J_i}$ é a Jacobiana da transformação projetiva para o ponto i.

Como o resultado é também uma Gaussiana, podemos aplicar um filtro passa-baixas, também Gaussiano, através da sua convolução com o *kernel* de reconstrução, gerando, assim, o *kernel* de reamostragem. O *kernel* do filtro de passa-baixas utilizado é:

$$g_{\mathrm{H}}(x) = \frac{1}{2\pi\sqrt{|\mathrm{H}|}} e^{-\frac{1}{2}x^T \mathrm{H}^{-1} x} \tag{2-6}$$

onde, normalmente, H é a matriz identidade 2x2.

A aplicação do *kernel* de reamostragem gera as contribuições do ponto no espaço de tela, com a cor do ponto ponderada pelo peso calculado no processo. O algoritmo de *z-buffering* é utilizado para restringir a acumulação gerada pela reamostragem, nas áreas visíveis (Gross07).

2.2

Origem dos Dados

Após aquisição e processamento dos dados sísmicos, os mesmos são armazenados e disponibilizados para visualização e interpretação através de *softwares* especialistas disponíveis no mercado.

Os dados utilizados neste trabalho foram gerados a partir de horizontes sísmicos interpretados manualmente em cada seção sísmica 2D, caso com grande representatividade na interpretação de horizontes. Estes dados foram interpolados na ferramenta GOCAD[1], utilizando *Discrete Smooth Interpolation* para a geração dos pontos que compõem a superfície que representa o horizonte sísmico em 3D, e exportados em arquivos XYZ, no formato ASCII, apenas com as coordenadas dos pontos, sem qualquer informação adicional.

Estes dados poderiam ter sido gerados das mais variadas formas, automatizadas ou manuais, e interpolados pelas mais variadas técnicas disponíveis. Contanto que a nuvem de pontos gerada representasse de forma satisfatória a superfície do horizonte interpretado, os passos e metodologias utilizados neste trabalho teriam igual aplicação. Entretanto, a abordagem utilizada no trabalho representa uma das práticas comumente adotadas por intérpretes na identificação de horizontes sísmicos.

2.3 Organização de Dados

Como mencionado na seção **1.2.2**, no *Surface Splatting*, para cada ponto é gerado um *splat*, com informações necessárias à sua renderização.

Neste trabalho, para cada *splat* armazenamos as seguintes informações, que são utilizadas pelo método:

Tabela 2.1: Estrutura do *splat* armazenado

Dado	Tipo	Tamanho (bytes)
Posição	float x 3	12
Normal	float x 3	12
Tamanho do splat	float	4
Cor	byte x 3	3

Uma característica do algoritmo de *Surface Splatting* é a não necessidade de ordem pré-definida para a reamostragem de cada *splat*. Assim, a organização dos *splats* poderia ser feita sem qualquer custo adicional, utilizando estruturas simples, como vetores ou listas, com baixo custo de criação e inserção, e acesso a dados em tempo constante.

[1]GOCAD é marca registrada da Earth Decision and Paradigm Geotechnology

Entretanto, se desejarmos implementar técnicas como LOD (*level of detail*) para ganhar performance em grandes volumes de dados, cuja implementação a baixo custo é uma importante qualidade das abordagens baseadas em pontos, devemos optar por estruturas hierárquicas para o armazenamento.

Outro fator que devemos levar em consideração, devido à característica dos dados de entrada analisados, é o cálculo das normais para cada ponto, que será discutido com mais detalhes adiante. O seu cálculo demanda o conhecimento da vizinhança de um ponto, o que deve ser provido pela estrutura de dados escolhida para uso.

Estruturas que atendem às características requeridas e que vêm sendo amplamente utilizadas em técnicas de visualização baseadas em pontos são as octrees e kd-trees (Park04, Pajarola04).

Octrees são estruturas de dados hierárquicas, usadas comumente na representação de dados em três dimensões, que se baseiam no princípio da decomposição recursiva do espaço. Na aplicação de *region octrees* para pontos, a decomposição é feita dividindo-se um bloco em 8 octantes, recursivamente, até que em cada bloco gerado não haja nenhum ou apenas um ponto (ou um pequeno conjunto de pontos). Esta subdivisão pode ser representada por uma árvore de grau 8, onde o nó raiz representa todo o objeto (Samet89)

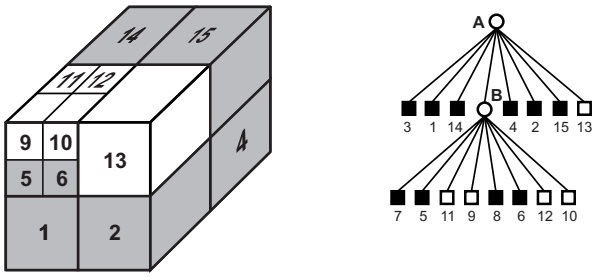

Figura 2.4: Exemplo de decomposição em blocos de uma octree e sua representação como árvore, onde as folhas brancas representam regiões sem pontos e as pretas regiões com pontos, adaptado de (Samet89)

A kd-tree é também uma estrutura hierárquica, criada em 1975 por Bentley (Bentley75) e, segundo Shakhnarovich (Shakhnarovich06), apesar de todos esses anos desde sua criação a kd-tree (e suas variantes) ainda é, provavelmente, uma das estruturas de dados mais utilizadas para buscas em espaços multidimensionais.

Para um conjunto de n pontos em um espaço d-dimensional uma kd-tree é gerada recursivamente da forma descrita a seguir. Inicialmente acha-se a mediana dos valores da i-ésima coordenada, começando por $i = 1$, gerando assim um valor M , de modo que os conjuntos de pontos que tem esta coordenada menor ou igual que M tem aproximadamente o mesmo número de elementos do conjunto de pontos com a esta coordenada maior ou igual a

M. Este conjunto de pontos P é então particionado em P_E e P_D, onde P_E é constituído pelos pontos com sua i-ésima coordenada menor ou igual a M (e P_D os maiores ou iguais). O processo é repetido recursivamente em P_E e P_D, com i substituído por $i + 1$ ou por 1, caso $i = d$.

Quando o conjunto de pontos em um nó tem apenas um ponto, a recursão é interrompida. O resultado deste processo é uma árvore binária, com n folhas e profundidade $\lceil \log n \rceil$ (Shakhnarovich06).

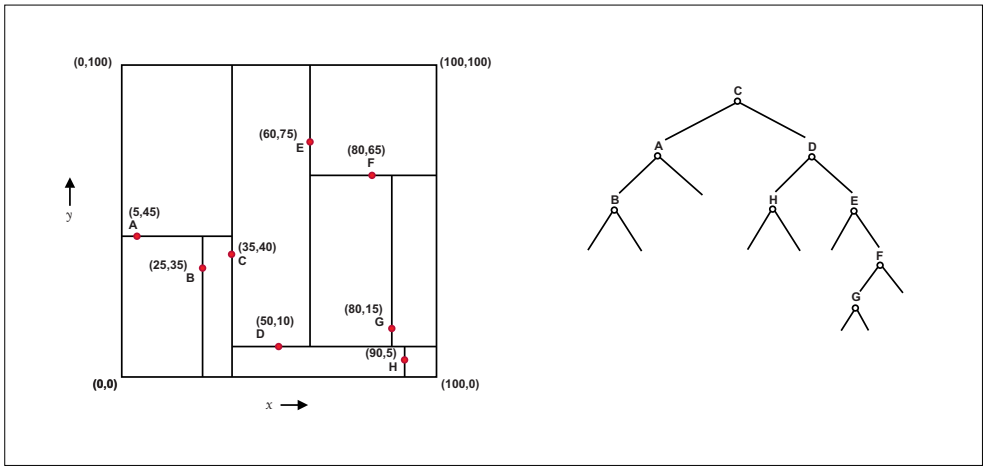

Figura 2.5: Exemplo de uma kd-tree e sua representação como árvore, adaptado de (Samet89)

Os pontos que representam as superfícies dos horizontes sísmicos interpretados, possuem características (como uma grande área, comportamento irregular e pequena espessura) que fazem com que a estratégia de decomposição utilizada pela octree gere um excessivo número de nós, gerando um *overhead* na sua utilização. Enquanto na kd-tree, opção adotada neste trabalho, a estrutura é mais compacta e a busca de vizinhos próximos bastante eficiente.

Outras estruturas com maior complexidade, como variantes e híbridas das mencionadas, poderiam ser utilizadas. Entretanto, o foco foi dado em métodos simples e eficientes, visando a possibilidade de implementação futura destas estruturas em placas gráficas.

A implementação utilizada no trabalho para a kd-tree, que armazena os *splats* da superfícies, foi a disponibilizada na biblioteca de algoritmos de geometria computacional CGAL [2].

[2]CGAL é o acrônimo para Computational *Geometry Algorithms Library*, uma biblioteca que implementa algoritmos robustos de geometria computacional para a academia e para a indústria. Seu desenvolvimento é mantido por uma comunidade de pesquisadores provenientes de vários institutos de pesquisa e empresas, em diversos países. Mais informações no site http://www.cgal.org.

2.4 Frustum Culling

Durante a visualização de uma área de interesse, que inclua um determinado horizonte sísmico, pode ocorrer de este horizonte não estar (ou estar apenas parcialmente) dentro da região visível ou *view frustum*, como exemplificado na figura 2.6.

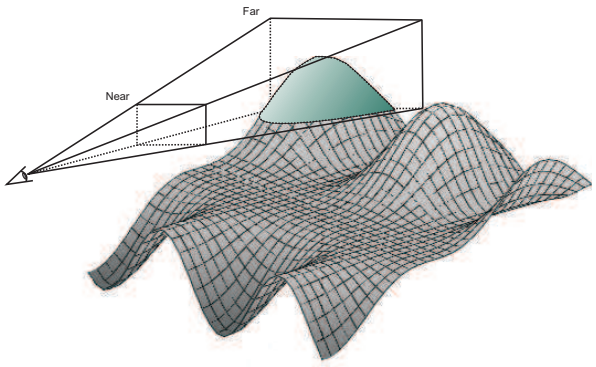

Figura 2.6: Exemplo de um *frustum* de visualização que contém apenas uma pequena parte de uma suprfície em sua região visível.

Para evitar que sejam efetuados os cálculos para visualização de *splats* que não estão na região visualizada pela câmera definida, pode ser aplicado o *frustum culling*, com o descarte destes *splats* antes que sejam encaminhados para o algoritmo de visualização.

Em cenas complexas, o uso de uma abordagem hierárquica para o *frustum culling* é de grande importância (Haines08). No caso da kd-tree, a estrutura hierárquica adotada, esta abordagem é iniciada na raiz. Primeiramente verificamos se o cubóide envolvente ou *bounding box* (BB) da raiz, que compreende todos os *splats* do horizonte, está dentro, fora ou faz interseção com o *frustum*. Se estiver dentro, então todos os *splats* estão na região visível e estes serão encaminhados para o algoritmo de visualização. Se estiver fora, então nenhum splat está na região visível e serão todos descartados.

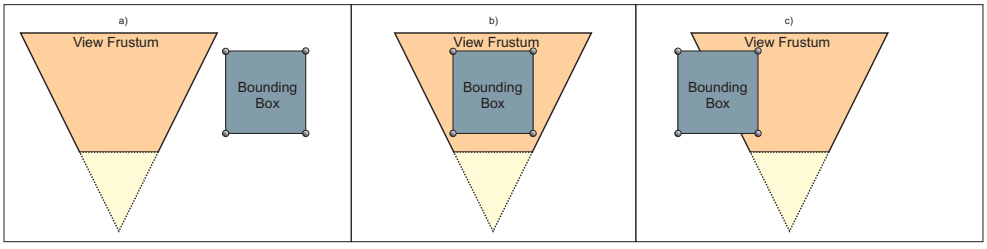

Figura 2.7: a) A *bounding box* está fora do *view frustum*; b) A *bounding box* está dentro do *view frustum*; c) A *bounding box* faz interseção com o *view frustum*.

Se a BB fizer interseção com o frustum, então devemos refazer o teste para as BB dos dois filhos, seguindo este algoritmo recursivamente até as folhas, como ilustrado na figura 2.8.

Nas folhas, também desenhamos os *splats* caso a BB faça interseção com o *frustum*.

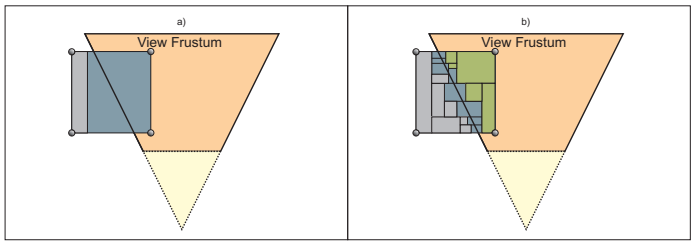

Figura 2.8: a) Após a primeira iteração: são verificadas as *bounding boxes* dos 2 filhos do nó raiz; b) Após diversas iterações, onde as regiões em cinza são estão fora do *frustum* (serão descartadas), as regiões verdes estão dentro (serão processadas) e as regiões em azul têm interseção com o *frustum* (serão recursivamente iteradas).

No algoritmo, inicialmente são obtidos os seis planos que contém as faces do *frustum*. Entretanto, como os pontos que geraram a kd-tree representam apenas os centros dos *splats*, poderiam haver descartes indevidos de *splats* cujos centros estivessem fora do *frustum*, embora parte do mesmo estivesse dentro (figura 2.9).

Para garantir que estes *splats* não serão desconsiderados, podemos aumentar o volume do *frustum* considerado, deslocando os planos de corte para fora em uma distância equivalente ao maior raio de *splat* utilizado. Assim, embora *splats* possam ser processados desnecessariamente, temos certeza de que nenhum *splat* será descartado indevidamente, o que poderia gerar erros na visualização.

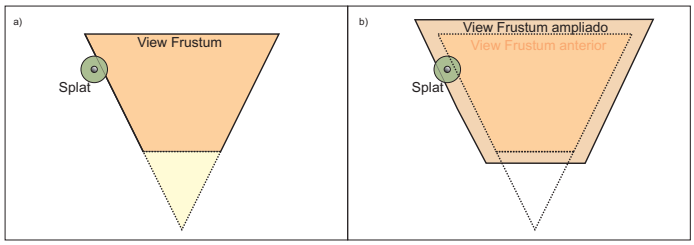

Figura 2.9: a) centro do *splat* está fora do *frustum*, mas *splat* deveria ser desenhado; b) Após apliação do *frustum*, centro do *splat* passa a estar dentro do *frustum*.

A obtenção dos seis planos de *frustum* na forma $ax + by + cz + d = 0$, com as normais direcionadas para seu interior, utilizando o OpenGL, pode feita diretamente a partir da matriz M = ModelviewProjection, na forma apresentada abaixo (a demonstração não será feita neste trabalho, mas pode ser consultada em (Gribb01)):

Tabela 2.2: Planos que delimitam o *frustum*

Plano	**Coeficientes**	**Plano**	**Coeficientes**
Esquerdo	$a = M_{41} + M_{11}$ $b = M_{42} + M_{12}$ $c = M_{43} + M_{13}$ $d = M_{44} + M_{14}$	Direito	$a = M_{41} - M_{11}$ $b = M_{42} - M_{12}$ $c = M_{43} - M_{13}$ $d = M_{44} - M_{14}$
Inferior	$a = M_{41} + M_{21}$ $b = M_{42} + M_{22}$ $c = M_{43} + M_{23}$ $d = M_{44} + M_{24}$	Superior	$a = M_{41} - M_{21}$ $b = M_{42} - M_{22}$ $c = M_{43} - M_{23}$ $d = M_{44} - M_{24}$
Frontal	$a = M_{41} + M_{31}$ $b = M_{42} + M_{32}$ $c = M_{43} + M_{33}$ $d = M_{44} + M_{34}$	Traseiro	$a = M_{41} - M_{31}$ $b = M_{42} - M_{32}$ $c = M_{43} - M_{33}$ $d = M_{44} - M_{34}$

Para aumentar o volume do *frustum* em uma distância r equivalente ao valor do maior raio de *splat* utilizado, precisamos, para cada plano, normalizar a equação obtida dividindo seus coeficientes pelo módulo do vetor (a,b,c) e somar ao novo componente d o valor do r obtido. Isto equivale a mover o plano na direção contrária à sua normal, no caso em direção à região externa ao *frustum*, de uma distância r.

A verificação de uma BB em relação ao *frustum*, citada anteriormente, utiliza a comparação de cada um de seus 8 pontos com cada plano do *frustum*. Se os 8 pontos estiverem na região externa de ao menos um plano, o BB está fora do *frustum*. Se os 8 pontos estiverem na região interna de todos os planos, o BB está dentro do *frustum*. Caso contrário o BB tem interseção com o *frustum*. Para determinarmos se um ponto está na região interna ou externa de um plano do *frustum*, podemos substituir os valores de x, y e z em $ax + by + cz + d$. Caso o resultado seja positivo, o ponto está do lado da normal no plano (lado interno do *frustum*, em relação ao plano), caso contrário o ponto está no lado de fora do volume de visualização.

Para maior detalhamento sobre os cálculos utilizados e sobre as técnicas de *frustum culling* consultar (Gribb01) e (Haines08).

2.5 Back Face Splats

Na visualização de superfícies baseadas em pontos, através do uso do *surface splatting*, temos que tratar o caso em que os *splats* não estão com a face da frente voltada para o observador. Neste caso, em aplicações usuais onde queremos visualizar apenas a casca externa da superfície, podemos descartar os *splats* que têm sua face de trás voltada para a câmera, em um processo conhecido como *back face culling*.

Entretanto, na visualização de horizontes sísmicos, estamos interessados em visualizar ambos os lados da superficie. Não podemos aplicar o *back face culling*, mas, ainda assim, precisamos endereçar o caso dos *splats* que não estão voltados para a câmera, pois estes *splats* possuem uma normal que aponta para o lado oposto ao do observador, o que afetaria o cálculo da iluminação.

Para resolver esta questão, identificamos os *splats* que estão com a face de trás voltada para a câmera e, para estes casos, consideraramo sua normal como a normal do *splat* invertida, apontando para o sentido oposto ao original.

A determinação de se um *splat* está apontando para a câmera ou não é feita a partir da análise do produto escalar da normal do *splat* pela direção da câmera (diferença entre a posição do *splat* e a posição da câmera).

$$n \cdot c_{dir} = \|n\| \, \|c_{dir}\| \cos\theta \tag{2-7}$$

onde $c_{dir} = (p_{splat} - p_{camera})$

Se $\cos\theta < 90°$, *splat* está virado para a câmera ($n \cdot c_{dir} > 0$)

Se $\cos\theta = 90°$, *splat* está perpendicular à câmera ($n \cdot c_{dir} = 0$)

$\cos\theta > 90°$, fundo do *splat* está virado para a câmera ($n \cdot c_{dir} < 0$)

Então, se $n \cdot c_{dir} < 0$, tratamos n como $-n$.

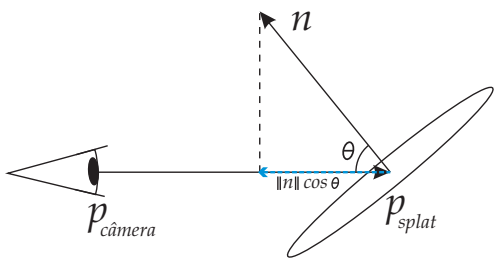

Figura 2.10: Identificação da face do *splat* voltada para a câmera.

2.6 Cálculo de Normais

Quando lidamos com horizontes rastreados diretamente em volumes sísmicos, onde o horizonte é na realidade parte do volume, realçado através da alteração de propriedades de visualização de *voxels* e funções de transferência, é de grande valor a utilização propriedades dos dados sísmicos para o cálculo das normais do horizonte, como em (PMario04), onde é apresentada a utilização do gradiente do campo escalar de fase instantânea, para este fim.

Entretanto, no caso estudado, onde as interpretações são processos manuais, baseados em uma análise visual dos dados e no conhecimento do intérprete, o uso de dados do volume sísmico para o cálculo das normais poderá gerar resultados incorretos. Por essa razão, este trabalho adota uma abordagem baseada na geometria da superfície gerada pela interpretação.

Neste caso, o problema se traduz na obtenção de normais em superfícies representadas por nuvens de pontos. E para esta questão existem inúmeras técnicas propostas em publicações da área, que variam bastante de acordo com a natureza dos dados e com o resultado final desejado.

Primeiramente podemos pensar na questão do ruído, comumente presente nos dados sísmicos registrados em uma aquisição. Técnicas para a sua atenuação, que podem ser utilizadas no processamento e mesmo no caso da visualização volumétrica, podem ser negativas para uma interpretação manual de um horizonte sísmico. Neste caso, informações geradas pelo intérprete seriam descartadas no processo e é desejável que a superfície interpretada seja visualizada da forma mais fiel possível à sua especificação pelo intérprete. Por isso, o ruído não foi levando em conta, na escolha do método de estimação de normal utilizado.

Duas abordadens podem ser destacadas. Uma abordagem numérica, que se baseia em técnicas de otimização. Outra, baseada em técnicas basicamente combinatórias, que aplicam alguma propriedade de Delaunay/Voronoi para a resolução do problema (Dey05).

Um dos métodos mais utilizados, que utiliza a abordagem de otimização, baseia-se na aplicação de mínimos quadrados na busca do melhor plano que se adapta à vizinhança do ponto (*plane fitting*) para o cálculo da normal. Inicialmente apresentado por Hoppe em (Hoppe92), com a utilização dos k vizinhos para a otimização, esta abordagem teve diversas variantes desenvolvidas, com alterações como a utilização de um raio ao redor do ponto, ao invés de k vizinhos e a utilização de ponderação pela distância dos vizinhos, apresentada em (Pauly03). Dey, em (Dey05), mostra em resultados comparativos, em que o método de *plane fitting* ponderado (figura 2.11) apresenta melhor performance

em comparação à implementação baseada no diagrama de Voronoi, apesar de ter menor precisão, especialmente no caso da presença de ruído. O autor conclui que nos casos em que não conhecemos a qualidade dos dados, a abordagem baseada em diagramas de Voronoi é mais segura, mas, caso contrário, deve-se optar pelos métodos de *plane fitting*.

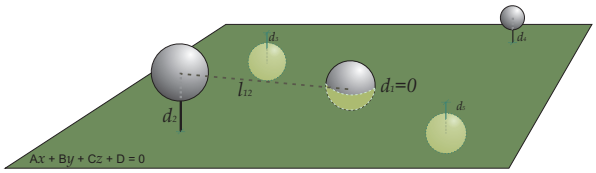

Figura 2.11: *Plane fitting* ponderado: busca de coeficientes do plano (A, B, C, D) que minimizam as distâncias dos pontos ao plano (d_1, d_2 , d_3 , ...), através de otimização por mínimos quadrados, ponderados pela distância ao ponto central. Quanto maior a distância do ponto ao ponto central (l_{12}, l_{13} , l_{14} , ...), menor o peso no cálculo. Coeficientes lineares do plano (A, B, C) compõem a aproximação da normal no ponto central d_1.

Um dos maiores problemas das técnicas baseadas em *plane fitting*, que afeta diretamente sua precisão, é a forma adequada para a escolha da vizinhança a ser considerada no cálculo da normal. Mitra, em (Mitra03), demonstra que para minimizar o erro no cálculo da normal onde não há ruído, devemos buscar uma área tão pequena quanto possível da vizinhança. Como, neste trabalho, não precisamos considerar o tratamento de ruído, devemos estabelecer a forma adequada de definir a menor vizinhança para uma boa estimação em nosso caso específico. Para isso, devemos analisar o comportamento dos dados utilizados.

No caso da interpretação manual, existe um comportamento que pode ser esperado do intérprete na identificação dos horizontes, em seções sísmicas. Em regiões de pouca curvatura, poucos pontos são utilizados, enquanto em regiões mais complexas, com grandes curvaturas, muitos pontos são inseridos pelo intérprete, visando um melhor delineamento da área interpretada. Podemos observar nos dados gerados que as áreas de maior curvatura possuem naturalmente maior densidade de pontos do que as áreas de menor curvatura. Assim, a utilização de um método baseado em um raio fixo a ser considerado para a vizinhança não é recomendado, pois para um raio pequeno, seriam privilegiadas áreas de maior curvatura, enquanto que um raio grande poderia trazer imprecisão nestas mesmas áreas.

A escolha de um número fixo de k-vizinhos, entretanto, resultará em uma área grande para regiões de baixa densidade, com pouca curvatura, e uma área pequena para regiões de grande curvatura, se mostrando uma opção válida para o dado utilizado. Devido ao não tratamento de ruído, devemos considerar o menor valor de k possível, para uma estimação precisa.

Se pensarmos no caso de uma distribuição regular de pontos, apenas oito pontos cobririam todas as direções ao redor do ponto desejado. Para um caso irregular qualquer, o número de pontos ideal para a vizinhança de um ponto pode variar para cada ponto, sendo impossível a escolha de um mesmo k ótimo para todos os pontos da superfície. Entretanto, apesar de o caso estudado ser de natureza irregular, a interpolação aplicada no processo possibilita uma distribuição mais uniforme dos pontos no espaço. Isso faz com que a aproximação de k para 8, como no caso regular, apresente bons resultados para os dados avaliados.

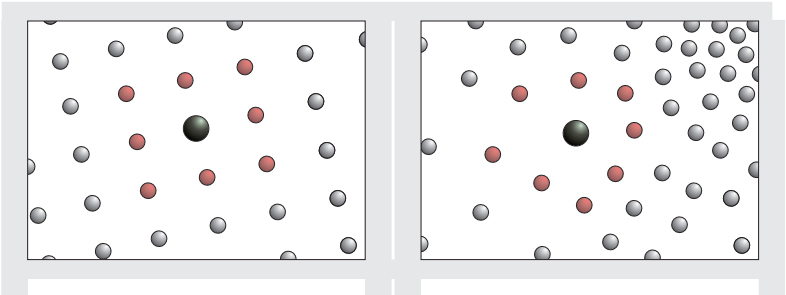

Figura 2.12: Ponto central e sua vizinhança. Na figura da esquerda, os pontos estão distribuídos de forma regular. Na figura da direita a distribuição é irregular.

Os passos adotados neste trabalho para o cálculo das normais do horizonte, são descritos a seguir.

Todos os pontos são varridos sem ordem pré-estabelecida, sendo que para cada ponto é efetuado o cálculo de sua normal.

O cálculo da normal para um determinado ponto inicia com a identificação dos seus oito vizinhos mais próximos. Para isso, é utilizado um algoritmo de k-NN (k nearest neighbors) para a kd-tree, que tem complexidade $\mathrm{O}\,(k \log n)$.

É aplicada a técnica de *plane fitting*, que identifica o plano que passa pelo ponto varrido e minimiza as distâncias dos pontos da vizinhança a este plano. Para achar o plano que melhor se encaixa na vizinhança, é utilizado o método de mínimos quadrados, ponderados pelas distâncias ao ponto principal. Este método busca os coeficientes do plano que minimizam as distâncias mencionadas. Os coeficientes encontrados representam a normal do mesmo, sendo esta a normal do ponto varrido, procurada pelo método.

Na implementação foi utilizada a rotina de k nearest neighbors, implementada pela biblioteca CGAL, e o cálculo de mínimos quadrados ponderados implementado pela biblioteca GSL [3].

[3]GSL é o acrônimo de GNU Scientific Library, uma biblioteca numérica livre, que implementa mais de 1000 funções de uma ampla gama de rotinas matemáticas. Mais informações no site http://www.gnu.org/software/gsl/

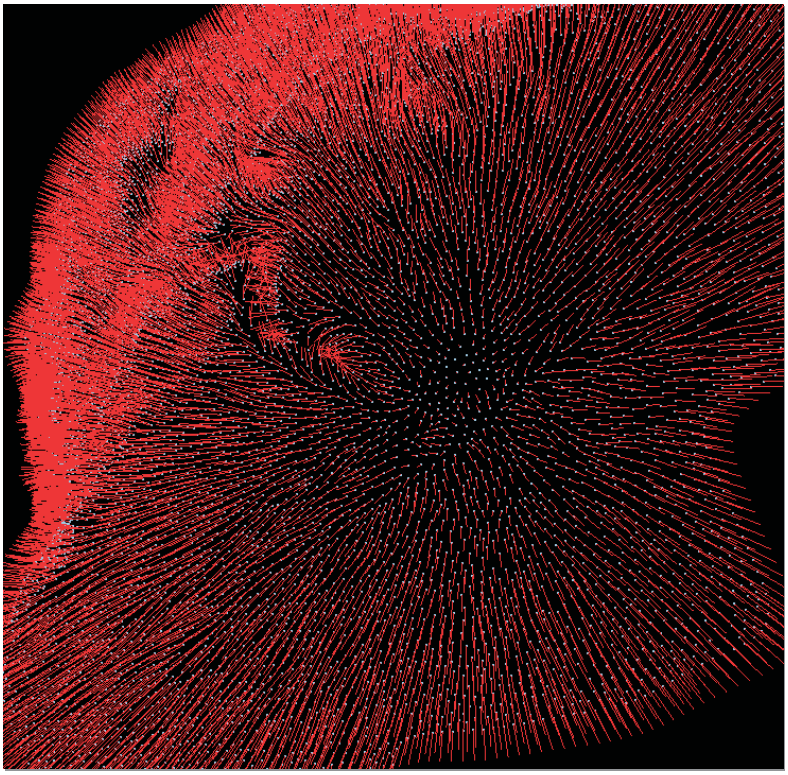

Figura 2.13: Normais calculadas para pontos de um horizonte sísmico interpretado

2.7 Estratégia de Iluminação

Antes de abordarmos a estratégia de iluminação empregada, é importante falarmos sobre a estrutura de *buffer* utilizada, capaz de acumular as contribuições de cada *splat* para a consolidação do *pixel*, em uma etapa posterior. O mecanismo escolhido para esse fim, como em (Rasanen02), foi o A-buffer.

O A-buffer, introduzido por Carpenter em (Carpenter84), é um *buffer* de acumulação para remoção de superfícies ocultas e *anti-aliasing*, onde cada *pixel* é composto por um conjunto de fragmentos que contribuem para o resultado final do mesmo. No caso da renderização de *splats*, cada fragmento representa uma amostra de um *splat* que contribui para a região definida pelo *pixel*.

O uso do A-buffer permite que os *splats* sejam desenhados em qualquer ordem, entretanto devemos estar atentos ao fato de que o A-buffer demanda uma quantidade de memória variável e sem um limite definido, pois não sabemos de antemão até quantos fragmentos pode ter um *pixel*. Esta pode ser uma informação relevante, uma vez que impõe restrições para a implementação desta estrutura em *hardware*.

Para a implementação da iluminação neste trabalho, duas abordagens foram adotadas. Uma na linha proposta originalmente por Zwicker (Zwicker01),

utilizando *deferred shading* para o cálculo da iluminação, e outra implementando a iluminação por *splat*, utilizada por (Rasanen02).

Na implementação do *deferred shading*, a equação de iluminação só é aplicada para o cálculo da cor no final do processo, utilizando as informações acumuladas do material e da normal para o pixel a ser renderizado.

Esta abordagem tem vantagens com relação ao processamento no caso de cenas muito complexas, onde existe uma parcela grande do modelo fora da área visível, pois só será aplicada a iluminação na região dos pixels gerados, além de gerar um resultado mais preciso. Entretanto, todos os dados necessários para a iluminação, como cor e normais, devem ser acumulados ponderadamente no A-buffer, o que demanda processamento e espaço em memória maiores.

Na implementação por *splat*, calculamos a iluminação para cada *splat*, em sua posição original. A cor gerada é acumulada no A-buffer para a geração da imagem final.

Neste caso, apenas as cores precisam ser acumuladas no A-buffer e menos dados precisam ser ponderados, tornando o processo mais ágil no caso geral, porém reduzindo a nitidez de imagens quando os *splats* estão próximos da câmera.

Ambas as estratégias foram testadas neste trabalho.

2.8 Algoritmo Implementado

Nesta seção é apresentado, passo a passo, todo o processo para a visualização de horizontes sísmicos, utilizado neste trabalho.

O processo começa com a carga do arquivo XYZ, que contém as posições dos pontos que compõem a superfície do horizonte sísmico interpretado.

Estes pontos são organizados em *splats* e armazenados em uma kd-tree.

Os *splats* são varridos e para cada um é calculada a normal, como detalhado na seção 2.6. Também neste ponto é calculado o tamanho do *splat*, utilizando como base para isso a distância do vizinho mais próximo na vizinhança do ponto, que se mostrou um bom estimador para os dados analisados.

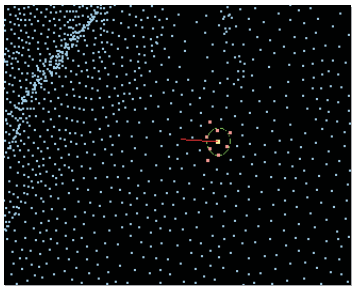

Figura 2.14: Para o *splat* central é encontrada a sua vizinhança, e então calculados o tamanho do *splat* e sua normal

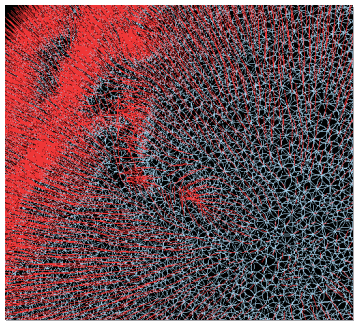

Figura 2.15: Tamanhos e normais de todos os *splats* de uma superfície

É, então, calculado o pré-filtro no espaço de tela homogêneo normalizado (para uso posterior no algoritmo), como:

$$\mathrm{V}'_h = \mathrm{S}\mathrm{V}_h\mathrm{S}^T \tag{2-8}$$

onde, $\mathrm{V_h}$ é a matriz variância do pré-filtro, que pode ser a matriz identidade, uma vez que não queremos maior atenuação de ruídos e S é a matriz:

$$\mathrm{S} = \begin{bmatrix} \frac{1}{\mathrm{Largura}_{\mathrm{viewport}}} & 0 \\ 0 & \frac{1}{\mathrm{Altura}_{\mathrm{viewport}}} \end{bmatrix} \tag{2-9}$$

Em seguida, são extraídos os planos que definem o *view frustum* para serem utilizados no *frustum culling* (ver seção 2.4).

O método adotado no *Surface Splatting* demanda dois passos para a sua execução. O primeiro varre todos os *splats*, calculando resultados até a sua rasterização, e o segundo varre todos os *pixels* do *frame buffer* colapsando os fragmentos acumulados em cada posição, para gerar a imagem final desejada.

No primeiro passo é feito o *fruatum culling*, onde todos os *splats* que não foram descartados por estarem fora da região visível são varridos, e para cada um são realizadas as atividades que serão definidas a seguir.

Para cada *splat* é verificado se a face do *splat* corrente voltada para a câmera é a sua face traseira, mas sem aplicar um *back face culling*, pois queremos visualizar os dois lados da superfície. Entretanto, ao visualizar o lado de trás, consideramos a normal invertida, para o cálculo da iluminação.

É gerada, então, a matriz M, que mapeia do plano do *splat* (plano parametrizado 2D) para o espaço de tela.

$$\mathrm{M} = \begin{bmatrix} \mathrm{t}_u \\ \mathrm{t}_v \\ \mathrm{p}_0 - u_0\mathrm{t}_u - v_0\mathrm{t}_v \end{bmatrix} \tag{2-10}$$

onde, $\mathrm{p}_0 = (\mathrm{p}_{0x}, \mathrm{p}_{0y}, \mathrm{p}_{0z})$ é a posição do *splat* corrente e (u_0, v_0) suas coordenadas correspondentes no plano parametrizado.

É obtida, em seguida, a matriz de variância do filtro de reconstrução, como:

$$\mathrm{V}_r = \mathrm{SS}^T \tag{2-11}$$

onde S é a matriz:

$$\mathrm{S} = \begin{bmatrix} \mathrm{s} & 0 \\ 0 & \mathrm{s} \end{bmatrix} \tag{2-12}$$

e s é o tamanho do *splat*.

A partir da matriz de variância, é calculada a matriz cônica 2x2, como:

$$\mathrm{Q} = (\mathrm{V}_r\mathrm{V}_r^T)^{-1} \tag{2-13}$$

É, então, calculada a matriz cônica 3 x 3, utilizando coordenadas homogêneas:

$$\mathrm{Q}_h = \begin{bmatrix} \mathrm{Q}_{11} & \mathrm{Q}_{12} & 0 \\ \mathrm{Q}_{21} & \mathrm{Q}_{22} & 0 \\ 0 & 0 & -\mathrm{F} \end{bmatrix} \tag{2-14}$$

onde, F é o tamanho do filtro de reconstrução. Assumimos, neste trabalho, $\mathrm{F} = 2$.

A partir de Q_h é calculada $\mathrm{Q'}_h$, a matriz cônica homogênea após o mapeamento projetivo:

$$\mathrm{Q'}_h = \mathrm{M}^{-1}\mathrm{Q}_h\mathrm{M}^{-1^T} \tag{2-15}$$

É, a seguir, calculado o *offset* central da cônica x_t e então derivada a matriz cônica, com aproximação afim $\mathrm{Q'''}$

Pode, então, ser derivada a matriz de variância da aproximação afim do filtro de reamostragem como:

$$V_\rho = V'_h + Q'''^{-1} \tag{2-16}$$

Invertendo a matriz de variância do filtro de reamostragem, obtemos a matriz cônica do filtro de reamostragem:

$$Q_\rho = V_\rho^{-1} \tag{2-17}$$

A partir de Q_ρ é possível encontrar o retângulo que contem a cônica no espaço de tela (*bounding rectangle*) e, então, fazer seu mapeamento para valores inteiros, representando as posições dos *pixels* no *frame buffer*.

Em seguida é feita a interpolação de profundidades e iniciada a rasterização.

Na rasterização, são varridos os pixels do *bounding rectangle*, verificando se cada um dos pontos varridos pertence à área ao *splat*. Caso pertença, são calculados o valor do peso gerado pela Gaussiana e a profundidade, sendo ambos acumulados no *frame buffer*. No caso do uso do *deferred shading*, a normal e a cor do *splat* também são acumulados, para posterior cálculo de iluminação. No caso do *shading* por *splat*, a iluminação é calculada neste momento e a cor, então, acumulada no *frame buffer*.

Como a aplicação da Gaussiana utiliza cálculo de exponencial, uma operação custosa em termos de processamento, e como sabemos que o domínio da função varia entre 0 e 2 (o tamanho do pré-filtro), foi criada uma tabela com valores pré-calculados, que é utilizada no momento da rasterização.

Encerrado o primeiro passo, o segundo é iniciado. Todos os *pixels* do *frame buffer* são varridos, executando-se as atividades descritas abaixo.

Para cada pixel, todos os fragmentos acumulados são ordenados de perto da câmera para longe.

São varridos todos os fragmentos, na ordem definida, sendo acumulados, ponderadamente ao peso armazenado, na cor final do *pixel*.

No caso do *deferred shading*, as normais também são acumuladas e ao final é efetivamente calculada a iluminação e determinada a cor do *pixel*.

É, então, criada uma textura, a partir do *buffer* final gerado, e esta textura é aplicada em um retângulo desenhado cobrindo toda a área de visualização.

É importante ressaltar que os valores gerados de profundidade são escritos no z-buffer, ao final do processo, de forma que a imagem gerada interaja da forma correta com outros objetos renderizados na cena. Este ponto tem

grande importância para o caso dos horizontes, que normalmente terão que ser visualizados junto a volumes sísmicos, poços, seções e outros objetos correlatos.