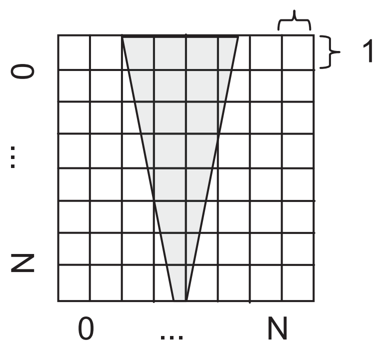
# 3.
# Técnicas de Nível de Detalhe

Numa visualização realística do oceano é imprescindível que o mesmo seja contínuo e que possa ser visto até o horizonte. Para isso, é necessário que ele possua uma malha que se estenda, pelo menos, do *Near Plane*, até o *Far Plane*. Entretanto, a discretização de uma malha que realize tal finalidade e, ainda, permita uma suave simulação do movimento oceânico, produz um elevado número de triângulos. Para exemplificar a situação, pode-se considerar o exemplo a seguir.

PUC-Rio - Certificação Digital Nº 0812608/CA

Dada uma câmera com os parâmetros [0.1 u.d.,1000.0 u.d.], sendo respectivamente (*Near Plane Distance*, *Far Plane Distance*), e um oceano com distância de amostragem de 1 u.d., o que permite uma simulação correta para ondas de até 0.5 u.d. de comprimento de onda (usando o Teorema de Nyquist[1]). Nessa situação, ilustrada na Figura 5, temos:




$$\Delta Camera = \Delta Ocean$$

$$Far - Near = (N - 0)\ \Delta D$$

$$1000.0 - 0.1 = N$$

$$N \cong 1000$$

**Figura 5:** Quantização de Amostras

Logo, seria necessário uma malha de 1000 x 1000 para representar o oceano, o que geraria cerca de 2 milhões de triângulos

---

[1] Nota-se que no caso de ondas há uma relação entre o comprimento de onda e a velocidade (Capítulo 4)

(usando a malha no espaço da câmera). Sendo assim, essa abordagem, também chamada de força bruta (*Brute Force*), não é adequada para a visualização, pois, mesmo que as placas atuais tivessem capacidade de processar essa quantidade de polígonos, o processo de renderização do oceano necessita de mais etapas de processamento, o que inviabiliza o emprego dessa técnica simplista.

Na área de computação gráfica, os terrenos são outro elemento natural que apresentam o mesmo problema. Tais elementos ainda possuem outro agravante: o armazenamento de informações na memória, sendo muitas vezes necessário o uso de *streaming* para acomodar tais informações. Entretanto, a visualização de oceanos, em situações habituais, não compartilha desse fato, sendo a única interseção o problema de resolução da malha necessária para representá-lo.

A forma empregada para a resolução dos problemas é a utilização das técnicas de nível de detalhe (*LoD*), que consistem em selecionar melhor qual resolução de informação é mais adequada a uma determinada distância, dessa forma, utiliza-se uma malha de maior resolução (maior número de triângulos) para partes do oceano que estão próximas do observador e uma malha de menor resolução (menor número de triângulos) para partes próximas do horizonte.

Tal funcionamento é justificado devido à transformação de projeção perspectiva. Essa, naturalmente, gera menos informação para elementos distantes da câmera. Tais técnicas também são utilizadas pelo hardware gráfico para o armazenamento e renderização de texturas (*Mip Maps*). A Figura 6 exemplifica a situação.

PUC-Rio - Certificação Digital Nº 0812608/CA

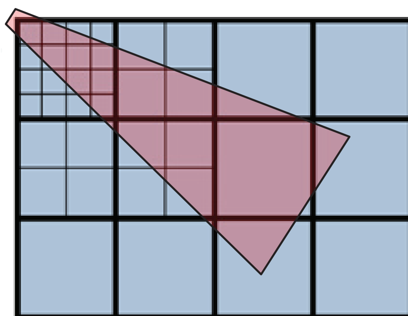


**Figura 6:** Funcionamento de *LoD*

Observa-se na Figura 6 que há uma variação do detalhamento com a distância, isso se deve ao algoritmo de *LoD* empregado.

O presente trabalho concentra-se apenas nas técnicas que utilizam a placa gráfica, adaptadas para a visualização de oceano em larga escala. Sendo assim, os algoritmos selecionados nesse contexto são *Radial LoD* (Kryachko, 2005), *GeoClipMap* (Asirvatham & Hoppe, 2005) e *Projected Grid* (Johanson, 2004).

Todas essas técnicas de *LoD* para o oceano concentram-se na geração da malha horizontal do oceano, não se preocupando com a variação vertical da mesma. No entanto, as técnicas podem utilizar duas estratégias de variação vertical para etapas seguintes, o cálculo procedimental diretamente no programa de vértice ou a utilização de mapas de altura, também chamados *displacement maps* ou *height maps*.

Os *height maps*(HM), ou também *height fields* (HF), representam uma função no espaço 2D, na qual, dado um ponto (s,t), esse mapa retorna a altura do mesmo. A implementação usual dos mesmos dar-se-á com uma representação matricial, logo, a função do HM é discretizada num domínio específico, p.e. $HM \in [0, N] \; x \; [0, N]$.

Para a obtenção de um resultado correto, deve-se tomar os devidos cuidados quanto a amostragem da função original, pois, sendo uma amostragem de um sinal (função), o HM deve obedecer ao Teorema

PUC-Rio - Certificação Digital Nº 0812608/CA

de Nyquist, a fim de fornecer informação suficiente para a reprodução mais fidedigna do mesmo. Todavia, a superfície oceânica deve ter aparência contínua e, se possível suave (derivável), logo, para isso, além da amostragem correta, é necessário o uso de uma função de interpolação. Contudo, o limite de Nyquist não garante a suavidade da curva interpolada.

Usualmente, utiliza-se a interpolação linear, pois, esta já está presente nativamente nos hardwares das placas gráficas, contudo, pode-se especificar interpoladores mais sofisticados. Observa-se que interpoladores lineares não garantem superfícies suaves, visto que o interpolador é apenas 1 vez derivável (veja a Figura 7).

Por fim, tendo um mapa de altura definido ($f_{HF}$) em um determinado subespaço discreto, para a obtenção da altura de um determinado ponto no espaço de simulação utiliza-se a expressão:

$$p_h(x, y) = p_{plane}\,(x, y) + f_{HF}(x, y) \cdot N_{plane}$$

**Equação 1:** Ponto de um Mapa de Altura em um plano

Na Equação 1, $p_{plane}$ é o ponto base do plano plane e $N_{plane}$ é a normal desse plano. Isso é necessário, pois, a maioria dos *HMs* possuem dados normalizados no intervalo [-1,1] ou [0,1].

Muitas vezes não é possível utilizar o HM em toda a área de simulação, dessa forma, pode-se utilizar as técnicas de repetição, ou *tiling*. O inconveniente dessas técnicas é a geração de padrões de repetição, os quais podem ser reduzidos com o uso de Perlin Noise (vide capítulo 4).

PUC-Rio - Certificação Digital Nº 0812608/CA

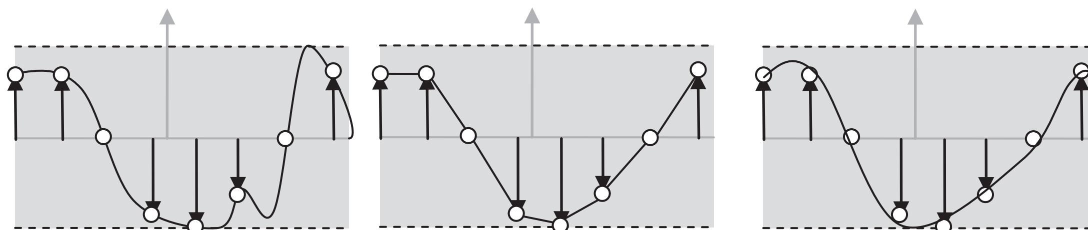
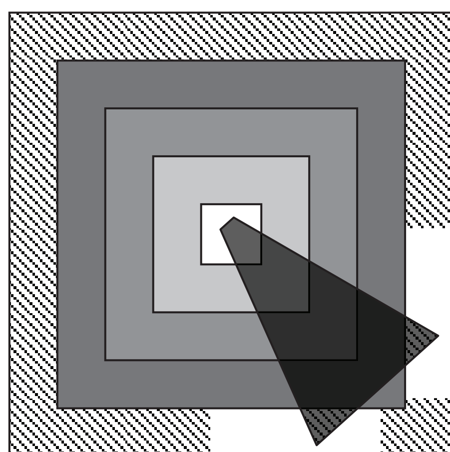


**Figura 7:** a) Função Original b) Interpolador Linear c) Interpolador Cúbico

## 3.1. GeoClipMap LoD

A primeira estratégia desenvolvida é a técnica de *GeoClipMap* (Asirvatham & Hoppe, 2005), ou *Geometry Clipmaps*, que é direcionada para o uso com a *GPU* e funciona oferecendo diferentes níveis de detalhamento de acordo com a distância do observador (dependente de visão). As vantagens dessa técnica para o oceano são a simplicidade e a capacidade de se realizar cortes (*culling*), reduzindo o número de vértices e fragmentos processados. O funcionamento principal da técnica pode ser visto na Figura 8 (sendo apresentados apenas os seis primeiros níveis).

PUC-Rio - Certificação Digital Nº 0812608/CA



| Nível | Dimensão | Cor |
|---|---|---|
| **0** | 512 x 512 | |
| **1** | 256 x 256 | |
| **2** | 128 x 128 | |
| **3** | 64 x 64 | |
| **4** | 32 x 32 | |
| **5** | 16 x 16 | |
| **6** | 8 x 8 | |
| **7** | 4 x 4 | |
| **8** | 2 x 2 | |

**Figura 8:** Anéis do GeoClipMap

O funcionamento geral consiste em renderizar retângulos com o centro recortado (também chamados de anéis), daí o nome *Clipmap*, pois recorta-se uma parte do mapa. Dessa forma, apenas a “borda” é desenhada. No exemplo da Figura 8, desenha-se cada resolução apenas

nas partes onde a cor for correspondente à resolução. Observa-se que a resolução de nível 0, não há um corte interno, isso devido à queda do tamanho do corte com o aumento do detalhamento.

Para o desenho de cada região de um determinado nível de detalhe são criados buffers estáticos. Esses são acomodados (redimensionando-os e transladando-os) nas posições correspondentes para a geração dos fragmentos. A técnica original utiliza uma série de malhas estáticas, chamadas de *footprints*, que podem ser vistas na Figura 9.a. Tal forma reduz o gasto de memória para armazenamento de vértices e índices, contudo, força o sistema a ter que fazer mais chamadas de desenho, cálculo de corte (*culling*) e trocas de contextos, além de não ser intuitivo. Isso é justificado no caso do *GeoClipMap* para terrenos, pois, nessa técnica não há *streaming*, todas as informações do terreno são carregadas de forma compactada na memória de vídeo, sendo descompactadas a medida da necessidade da informação e em nível de detalhes diferentes (Figura 9.b), ou seja, gasta-se muita memória da placa de vídeo para armazenar os dados do terreno.

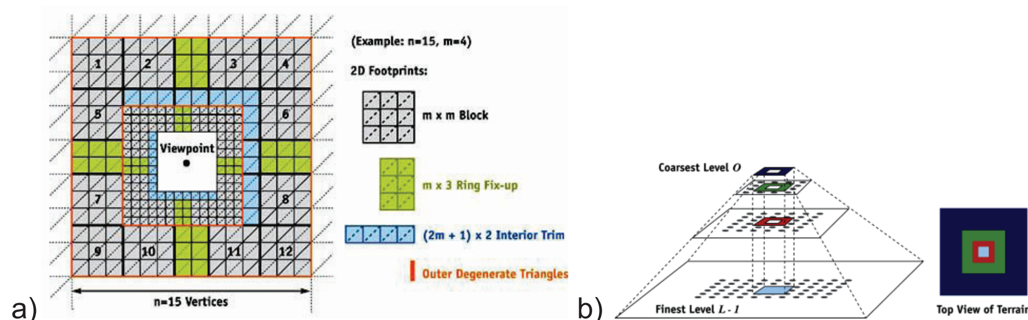



**Figura 9:** GeoClipMap (Asirvatham & Hoppe, 2005)

No entanto, a técnica para o uso com o oceano, em casos gerais, não necessita de grandes armazenamentos de dados na memória da placa de vídeo, visto que, na maioria dos casos, as informações serão geradas em tempo real, numa metodologia procedimental (procedural).

Assim, uma alternativa de desenvolvimento é a utilização do recurso de *Geometry Shaders*, o qual foi introduzido recentemente no

PUC-Rio - Certificação Digital Nº 0812608/CA

pipeline gráfico dos hardware moderno. Uma de suas funcionalidades é a geração de primitivas na própria placa, sendo assim, ao se enviar uma primitiva pode-se receber como resultado, várias outras primitivas. O efeito desse recurso na prática é permitir, por exemplo, que o desenho de um vértice possa gerar uma cena inteira. No entanto, diversos experimentos (Santos & Celes, 2009) mostram que o uso de *geometry shaders* apresentam uma baixa performance.

Outras investigações indicam que esse problema advém dos atual recusos do *hardware* gráfico. A maioria da *GPUs* implementam essa funcionalidade com diversas restrições, como, por exemplo, muitas *GPUs* não são capazes de escrever mais de 1024 valores escalares num mesmo *geometry shader*, isso limita a saída de apenas 340 vértices. Além disso a exportação de mais de uma dúzia de primitivas resulta em uma severa degradação da performance. Assim, o uso dos geometry shaders não foi explorado no presente trabalho.

Dessa forma, a adaptação efetuada da mesma consiste na criação uma série de *buffers* de vértices estáticos, em progressão geométrica de base 2, que são utilizados de acordo com o detalhe necessário. As dimensões podem ser vistas na tabela da Figura 8.

O nível de maior detalhe possue um espaçamento entre vértices de ΔD unidades de distância. Para as direções horizontais, a cada redução de resolução essa distância é duplicada em relação à mesma praticada na resolução imediatamente superior, ou seja, $\Delta D_{x+1} = \begin{cases} 2\,\Delta D_x, se\ x\ \neq 0 \\ \Delta D, se\ x = 0 \end{cases}$, onde x é o nível. A Figura 10 ilustra a renderização (no caso ilustrado 5 níveis).

PUC-Rio - Certificação Digital Nº 0812608/CA

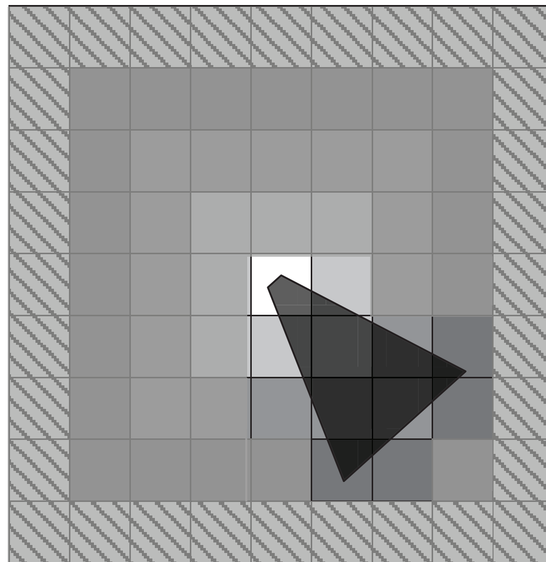


**Figura 10:** Funcionamento do *GeoClipMap*

PUC-Rio - Certificação Digital Nº 0812608/CA

Os pedaços acinzentados estão fora do campo de visão, dessa forma, podem ser descartados. Para a implementação dessa técnica, pode-se dividi-la em duas etapas: A etapa de geração dos buffers e especificação das posições e escalas, e a etapa de desenho propriamente dito.

Com essas adaptações, percebe-se que cada nível gera sempre menos vértices que o anterior, pois, sendo:

$$Size(i) = \begin{cases} k, & i \in \mathbb{N}, \quad i = 0 \\ \dfrac{Size(i-1)}{2}, & i \in \mathbb{N}, \quad i > 0 \end{cases}$$

**Equação 2:** Equação de recorrência do tamanho das grades

Resolvendo a equação de recorrência (Equação 2) (Cormen et al., 2001), temos:

$$Size(i) = \frac{k}{2^i}$$

**Equação 3:** Tamanho da grade

Daí:

$$N_i = [Size(i)]^2$$

**Equação 4:** Número de Vértices da grade

Além disso, cada anel possui dimensão dada por:

$$D_i = \begin{cases} 1, & i \in \mathbb{N}, \quad i = 0 \\ D_{i-1} + 2, & i \in \mathbb{N}, \quad i > 0 \end{cases}$$

$$D_i = D_0 + i \cdot 2 = 1 + i \cdot 2$$

**Equação 5:** Dimensão de Anel

Seguindo, como o total para cada anel pode ser dado pela expressão:

$$T_i = (D_i^2 - D_{i-1}^2) \cdot N_i = (D_0 + i \cdot 2)^2 \cdot Size(i)^2$$

$$T_i = \{(1 + i \cdot 2)^2 - [1 + (i-1) \cdot 2]^2\} \cdot \left(\frac{k}{2^i}\right)^2$$

$$T_i = \begin{cases} k^2, & i \in \mathbb{N}, \quad i = 0 \\ 8 \cdot i \cdot \left(\frac{k}{2^i}\right)^2, & i \in \mathbb{N}, \quad i > 0 \end{cases}$$

**Equação 6:** Número de Vértices por Anel

Assim, a variação entre dois anéis consecutivos é expressa por:

$$\Delta T_i - \Delta T_{i-1} = 8 \cdot i \cdot \left(\frac{k}{2^i}\right)^2 - 8 \cdot (i-1) \cdot \left(\frac{k}{2^{i-1}}\right)^2$$

$$= 8 \cdot \left(i \cdot \left(\frac{k^2}{4^i} - \frac{4k^2}{4^i}\right) - \frac{4k^2}{4^i}\right) = -\frac{8k^2}{4^i}(3i + 4) < 0$$

Logo, apesar de ser necessário desenhar mais grades a cada redução do nível de detalhe, a queda exponencial reduz significativamente o número de vértices necessários, dessa forma, a diferença é negativa.

O processo de geração dos dados pode ser desenvolvido seguindo o algoritmo proposto na Tabela 1:

**Tabela 1:** Algoritmo de Geração do GeoClipMap (Oceano)

```
01 void SetUpVertices(int nBuffers, int Levels, int Size)
02 {
03     VECTOR3* vertices;
04     short* indices;
05
06     VertexBuffer** mVertexBuffer = new VertexBuffer*[nBuffers];
07     IndexBuffer** mIndexBuffer = new IndexBuffer*[nBuffers];
08     int mSizes = new int[nBuffers];
09
```

PUC-Rio - Certificação Digital Nº 0812608/CA

```
10      int iSize = Size;
11      float fFactor = 1;
12
13      for (int i = 0; i < nBuffers; i++)
14      {
15          fFactor = Size / (float)(iSize);
16
17          // Create the vertex buffer
18           CreateVertexBuffer(iSize * iSize * sizeof(VECTOR3),
&mVertexBuffer[i]);
19
20          // Fill the vertex buffer.
21          m_pVertexBuffer[i]->Lock(0,iSize*iSize*sizeof(VECTOR3),
(void**)&vertices);
22
23          for (int x = 0; x < iSize; x++)
24          {
25              for (int y = 0; y < iSize; y++)
26              {
27                  vertices[x + y * iSize].x = x * fFactor;
28                  vertices[x + y * iSize].y = 0;
29                  vertices[x + y * iSize].z = y * fFactor;
30              }
31          }
32
33          mVertexBuffer[i]->Unlock();
34
35          vertices = NULL;
36
37          CreateIndexBuffer((iSize-1)*(iSize-1)*6*sizeof(short),
&mIndexBuffer[i]);
38          mIndexBuffer[i]->Lock( 0, 0, (void**)&indices);
39
40          mSizes[i] = (iSize - 1) * (iSize - 1) * 6;
41
42          int counter = 0;
43          for (int y = 0; y < iSize - 1; y++)
44          {
45              for (int x = 0; x < iSize - 1; x++)
46              {
47                  int lowerLeft = x + y * iSize;
48                  int lowerRight = (x + 1) + y * iSize;
49                  int topLeft = x + (y + 1) * iSize;
50                  int topRight = (x + 1) + (y + 1) * iSize;
51
52                  indices[counter++] = topLeft;
53                  indices[counter++] = lowerRight;
54                  indices[counter++] = lowerLeft;
55
56                  indices[counter++] = topLeft;
57                  indices[counter++] = topRight;
58                  indices[counter++] = lowerRight;
59              }
60          }
61
62          mIndexBuffer[i].Unlock();
63
64          iSize = iSize / 2;
65
66          vertices = NULL;
67          indices = NULL;
68      }
69
70      mToroildalPosition = new VECTOR3*[Levels];
71
72      int bufferlevel[] = { 0, 1, 1, 2, 2, 2, 2 };
73
74      mToroildalPosition[0] = new VECTOR3[1];
75      mToroildalPosition[0][0] = VECTOR3(0, 0, 0);
76
77      for (int i = 1; i < Levels; i++)
78      {
79          mToroildalPosition[i] = new VECTOR3[8 * i];
80
```

PUC-Rio - Certificação Digital Nº 0812608/CA

```
81          int len = 1 + 2 * i;
82          int sub = 0;
83          int center = len / 2;
84          for (int x = 0, w = 0; x < len; x++)
85          {
86              for (int z = 0; z < len; z++, w++)
87              {
88                  if ((x > 0 && x < len - 1) && (z > 0 && z < len - 1))
89                  {
90                      sub -= 1;
91                      continue;
92                  }
93
94                mToroildalPosition[i][w+sub]=VECTOR3((x-center)*Size,
bufferlevel[i], (z - center) * Size);
95              }
96          }
97        }
98 }
```

O algoritmo, escrito em linguagem pseudo-C, descreve uma forma de geração automática dos *buffers* e computação do tamanho e posição dos mesmos. As linhas de 6 a 8 criam vetores para armazenar os dados, observa-se que o número de *buffers* a serem criados é passado no argumento *nBuffers*. A cada *buffer* é computado o tamanho [sempre metade do *buffer* anterior (linha 64)] e criado o *buffer* da malha discretizada. A variável *fFactor* atua como uma escala para a amostragem. As linhas 37 a 62 concentram-se na criação do *buffer* de índices da malha.

O cálculo do posicionamento dos pedaços é computado nas linhas 70 a 97. Observa-se que o primeiro nível é calculado fora do laço iterativo da linha 77, conforme a Equação 6, e os demais computados iterativamente. Para o cálculo do posicionamento, calcula-se a dimensão do nível (dada pela Equação 5) e computa-se o centro do anel (metade da dimensão). Então se itera pelo tamanho do anel (tipicamente um caminhamento em uma matriz) analisando apenas os elementos de borda (linha 88), daí apenas recomputa-se o índice no vetor de posições (linha 94) e calcula-se a posição.

A fim de aproveitar a estrutura do vetor 3D, utiliza-se na coordenada vertical a informação do nível do anel. Observa-se que na linha 72 são definidos os níveis de *buffer*, isso permite repetir níveis para

PUC-Rio - Certificação Digital Nº 0812608/CA

melhor visualização do oceano. Na implementação final isso é passado como parâmetro e deve possuir *nBuffers* elementos.

O processo de renderização pode ser realizado seguindo o algoritmo proposto na Tabela 2.

**Tabela 2:** Algoritmo de Desenho GeoClipMap

```
01 void Draw(int nBuffers, int Levels, int Size)
02 {
03       ...
04
05           int stride = GeoClipMapLodManager::getSingleton()->Stride();
06        for (register int i = 0; i < GeoClipMapLodManager::Levels ; i++)
07        {
08            int Size = 0;
09            int BufferSize = 0;
10            int LastPlane = 0;
11     VECTOR3* Walk = GeoClipMapLodManager::getSingleton()->getPosition(i, Size);
12
13            for (int j = 0; j < Size; j++)
14            {
15                int BufferLevel = (int)Walk[j].y;
16
17                GridPos[0] = Walk[j].x;
18                GridPos[1] = Walk[j].z;
19
20               CullingPosition=pos-GeoClipMapLodManager::getSingleton()-
>HalfSize();
21                CullingPosition.y = 0;
22                CullingPosition.x += GridPos[0];
23                CullingPosition.z += GridPos[1];
24
25              if(FrustumCulling::getSingleton()->Intersect(
GeoClipMapLodManager::getSingleton()->Min(), GeoClipMapLodManager::getSingleton()-
>Max(), CullingPosition, LastPlane) )
26                {
27                    mEffect.setValue("GridPos", GridPos);
28                    VertexBuffer* Buffer =
GeoClipMapLodManager::getSingleton()->getVertexBuffer(BufferLevel);
29                    IndexBuffer* ibo = GeoClipMapLodManager::getSingleton()-
>getIndexBuffer(BufferLevel, BufferSize);
30
31                    // SET BUFFERS
32                    SetVertexBuffer( 0, Buffer, 0, stride );
33                    SetIndices(ibo);
34
35                    DrawIndexedPrimitive(TRIANGLELIST, BufferSize/3 );
36                    mRenderedTiles++;
37                }
38            }
39        }
40     ...
41 }
```

O código da Tabela 2 apresenta a forma de realizar as chamadas de desenho para o algoritmo de *GeoClipMap*. O funcionamento é simples, itera-se por todos os níveis tomando o vetor de posições dos pedaços e, então, realiza-se outra iteração sobre o número de pedaços do nível (obtido na linha 11). Após isso, calcula-se a posição final e testa-se a

PUC-Rio - Certificação Digital Nº 0812608/CA

pertinência do pedaço no volume de visão do observador. Sendo esse pertinente ao volume, ele é desenhado e sua posição é passada como parâmetro do programa de *GPU*.

O grande problema dessa técnica é a geração de *cracks*, ou ranhuras, nas fronteiras de níveis de resolução diferente (figura 11). Isso é contornado utilizando transições de morfologia, ou uma cor padrão para cobrir tais falhas.

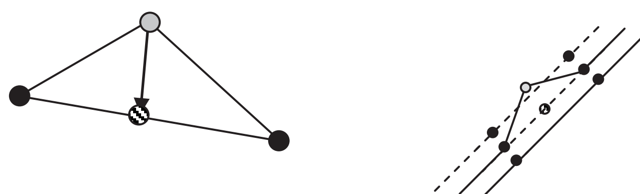


**Figura 11:** Formação de Ranhuras

O efeito ocorre devido à redução abrupta do número de vértices nas fronteiras dos níveis de detalhe. A morfologia aplicada é muito simples, bastando estimar o valor que o vértice de menor resolução estaria recebendo.

Na Figura 11, os vértices de cor preta existem em ambos os níveis, contudo o vértice de cor cinza só existe no nível de maior detalhe. Como a ligação entre vértices é feita por uma aresta linear, para estimar a posição que o vértice de cor cinza deve assumir, para corrigir a falha, deve-se apenas calcular o ponto médio entre os vértices de cor preta, gerando o vértice achurado.

Por fim, essa técnica ainda apresenta uma grande vantagem em relação às outras, pois, sendo ela utilizada para a visualização de terreno, pode-se realizar uma visualização conjunta de oceano e terreno de forma integrada. Isto diminui o consumo de memória e facilita a passagem de informações do terreno para o oceano, como, por exemplo, informação de profundidade.

PUC-Rio - Certificação Digital Nº 0812608/CA

## 3.2. Radial Grid Lod

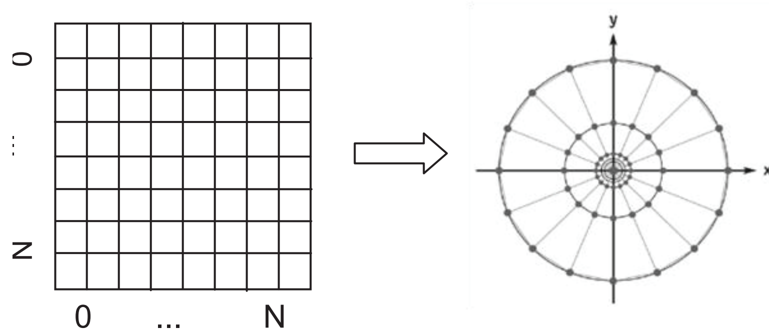



**Figura 12:** Transformação do Radial LoD

A técnica de *Radial LoD* (Kryachko, 2005), ou Nível de Detalhe Radial, é uma das técnicas mais usadas para *LoD* de oceano, pois, trata-se de uma técnica simples, de baixo custo e capaz de ser totalmente implementada na *GPU*, podendo, também, ser totalmente pré-computada.

O grande benefício dessa técnica está no fato de gerar mais vértices perto do observador e menos vértices, longe; sendo, portanto, uma técnica dependente de visão (Figura 12). A computação das posições dos vértices cria uma malha tecelada (discretizada) de dimensão (N x M) e utiliza-se das seguintes equações para gerar as novas posições em expansão radial:

$$r = a_0 + a_1 i^k$$
$$x_{i,j} = r\,cos\left(\frac{2\pi j}{M}\right)$$
$$y_{i,j} = r\,sin\left(\frac{2\pi j}{M}\right)$$

**Equação 7:** Equações do *Radial LoD*

Onde $\boldsymbol{i} \in [\boldsymbol{0}, \boldsymbol{N}-\boldsymbol{1}], \boldsymbol{j} \in [\boldsymbol{0}, \boldsymbol{M}-\boldsymbol{1}]$ e $a_0$ e $a_1$ são escolhidos da seguinte maneira:

PUC-Rio - Certificação Digital Nº 0812608/CA

$$r_0 = a_0 = distancia_minima$$
$$r_{N-1} = a_0 + a_1(N-1)^k = distancia_maxima$$

**Equação 8:** Escolha dos parâmetros do *Radial LoD*

Dessa forma, utilizando essas expressões, pode-se definir um oceano variando do *near plane* ao *far plane*. O parâmetro k e a dimensão da malha são fundamentais para obtenção de uma visualização de qualidade. Nesse sentido, quanto maior o valor de M e N, maior será a qualidade da visualização para um mesmo k, pois, quando há poucos vértices disponíveis, a visualização dos pontos distantes apresenta um prejuízo visual muito grande, gerando artefatos e incoerências na simulação, devido à baixa amostragem.

O parâmetro k define o grau de tecelagem da malha, quanto maior o valor, mais pontos serão gerados próximos ao observador e, conseqüentemente, menos longe do mesmo. A implementação original (Kryachko, 2005) utiliza o valor de k como sendo 4, no entanto, esse valor mostrou-se muito exagerado, pois, força a tecelagem de uma malha muito coesa próximo do observador e muito espaçada longe, gerando muitos artefatos próximo ao *far plane*. Dessa forma, optou-se pelo expoente quadrático por distribuir melhor os vértices e, assim, evitar problemas de artefatos. Contudo, o valor é de livre escolha, de modo que o objetivo é reduzir ao máximo os artefatos gerados.

Outro problema desse algoritmo é o fato de não ser possível utilizar técnicas de descarte de geometria (*culling*). Isso se deve à forma de geração da malha radial. Essa falha é intrínseca ao método, ou seja, não é possível contornar o problema.

Quanto aos artefatos gerados, esses podem ser reduzidos com uma maior dimensão da malha tecelada, além da utilização de técnicas de *Depth of Field* (campo de profundidade), que suavizam as regiões mais distantes e névoa. Todavia, isso é um truque visual para reduzir o problema.

PUC-Rio - Certificação Digital Nº 0812608/CA

O processo de geração dos dados pode ser desenvolvido seguindo o algoritmo proposto na Tabela 3:

**Tabela 3:** Algoritmo de Geração do *Radial LoD*

```
01 void RadialLodManager::SetUpVertices()
02 {
03     VECTOR3* vertices;
04     short* indices;
05     VertexBuffer* mVertexBuffer = NULL;
06     IndexBuffer *mIndexBuffer = NULL;
07
08     // Create the vertex buffer.
09     CreateVertexBuffer( Size * Size * sizeof( VECTOR3), &mVertexBuffer);
10
11       m_pVertexBuffer->Lock( 0, Size * Size * sizeof( VECTOR3 ));
12
13     // Radial Data
14     double MaxCoef = (mMaxDistance - mMinDistance) / pow(Size - 1, mExponent);
15     for (register int x = 0; x < Size; x++)
16     {
17         for (register int y = 0; y < Size; y++)
18         {
19             float i = x;
20             float j = y;
21             double r = mMinDistance + MaxCoef * pow(i, mExponent);
22
23            vertices[x + y * Size].x = (float)(r*cos(2.0*M_PI*j/ (Size - 1)));
24            vertices[x + y * Size].y = 0;
25            vertices[x + y * Size].z = (float)(r*sin(2.0*M_PI*j/ (Size - 1)));
26         }
27     }
28
29     mVertexBuffer->Unlock();
30
31     vertices = NULL;
32
33     CreateIndexBuffer((Size-1)*(Size-1)*6*sizeof( short ), &mIndexBuffer);
34
35     mIndexBuffer->Lock( 0, 0, (void**)&indices);
36
37     m_BufferSize = (Size - 1) * (Size - 1) * 6;
38
39     int RadialCounter = 0;
40     for (int y = 0; y < Size - 1; y++)
41     {
42         for (int x = 0; x < Size - 1; x++)
43         {
44             int lowerLeft = x + y * Size;
45             int lowerRight = (x + 1) + y * Size;
46             int topLeft = x + (y + 1) * Size;
47             int topRight = (x + 1) + (y + 1) * Size;
48
49             indices[RadialCounter++] = topLeft;
50             indices[RadialCounter++] = lowerRight;
51             indices[RadialCounter++] = lowerLeft;
52
53             indices[RadialCounter++] = topLeft;
54             indices[RadialCounter++] = topRight;
55             indices[RadialCounter++] = lowerRight;
56         }
57     }
58
59     mIndexBuffer->Unlock();
60     indices = NULL;
61
62 }
```

PUC-Rio - Certificação Digital Nº 0812608/CA

A geração da malha do *Radial LoD* (Tabela 3) é bastante similar à do *GeoClipMap*, contudo, não é necessário criar vários *buffers*. Dessa forma, cria-se apenas o *buffer* principal com as coordenadas já recalculadas (linhas 14 a 27). O processo de computação dos índices é idêntico ao do *GeoClipMap*. O processo de renderização pode ser realizado seguindo o algoritmo proposto na Tabela 4.

**Tabela 4:** Algoritmo de Desenho do *Radial LoD*

```
01 void Draw(int nBuffers, int Levels, int Size)
02 {
03       ...
04
05    int stride = RadialLodManager::getSingleton()->Stride();
06    int BufferSize = RadialLodManager::getSingleton()->BufferSize();
07
08    VertexBuffer* Buffer = RadialLodManager::getSingleton()->getVertexBuffer();
09    IndexBuffer* ibo = RadialLodManager::getSingleton()->getIndexBuffer();
10
11    // SET BUFFERS
12    SetVertexBuffer ( 0, Buffer, 0, stride );
13    SetIndices(ibo);
14
15    DrawIndexedPrimitive( TRIANGLELIST, 0, 0, BufferSize, 0, BufferSize / 3 );
16       ...
17
18 }
```

A Tabela 4 mostra o algoritmo de desenho do *Radial LoD*. Observa-se uma simplificação em relação ao do *GeoClipMap*. Isso ocorre devido ao fato de não ser necessária a utilização de vários *buffers* e não ser realizado o teste de corte de volume de visão. Contudo, é possível realizar testes de corte se a malha radial for construida como fatias.

## 3.3. Projected Grid Lod

A técnica do *Projected Grid* (Johanson, 2004), ou Grade Projetada, baseia-se no modo como são feitas as transformações de espaços vetoriais de um sistema de rasterização como o OpenGL e Microsoft DirectX. A ideia desse algoritmo é descrever uma grade no espaço pós-perspectivo (espaço após a projeção perspectiva, ou espaço projetivo) e projetá-la sobre um plano no espaço vetorial do mundo. Dessa forma, pode-se definir a seguinte sequência de passos no algoritmo:

- Criar uma grade discretizada no espaço pós-perspectivo que seja ortogonal ao observador.

PUC-Rio - Certificação Digital Nº 0812608/CA

- Projetar a grade sobre um plano no espaço do mundo.
- Modificar grade.
- Desenhar grade.

Observa-se que se não houver uma alteração na grade no espaço do mundo, a mesma será desenhada com um plano paralelo ao observador (conforme a definição da mesma). O modelo de definição do algoritmo pode ser visto na Figura 13:

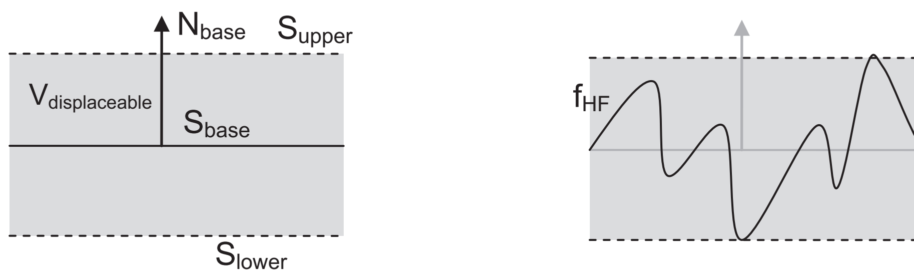



**Figura 13**: Modelagem do *Projected Grid*

Na figura há três planos de controle, o plano base ($S_{base}$), o plano limite superior ($S_{upper}$) e o plano limite inferior ($S_{lower}$). O volume gerado pelos planos $S_{upper}$ e $S_{lower}$ ($V_{displaceable}$) é o volume no qual a função do mapa de altura agirá. Ele é o *bounding volume*, ou volume de envolvente, da malha do oceano.

Dessa forma, pode-se expressar tais planos como:

$$p_{upper} = p_{base} + N_{base} * \max(f_{HF}), N_{upper} = N_{base}$$
$$p_{lower} = p_{base} - N_{base} * \min(f_{HF}), N_{lower} = N_{base}$$

**Equação 9:** Expressão dos planos de controle do Projected Grid

PUC-Rio - Certificação Digital Nº 0812608/CA

Na Equação 9, $f_{HF}$ (x,y) é a função contínua[2] do mapa de altura, definida em [0,N-1] x [0,N-1], sendo N a dimensão, ou número de amostras (pontos), da malha.

Sabendo que $f_{HF}$ é uma função limitada, já que é definida apenas no espaço da malha, certamente existe um mínimo global e um máximo global. Assim, é possível definir os planos de controle.

A transformação da grade do espaço do mundo para o espaço pós-perspectivo é feita através da matriz de mudança de base gerada pelas matrizes de visão (*View*) e projeção (*Projection*). Dessa forma, a transformação de um ponto no espaço do mundo para o espaço pós-perspectivo é realizada pela expressão:

$$p_{pos} = (M_{View} \cdot M_{Projection}) * p_{world}$$

**Equação 10:** Expressão do ponto projetado (Pós-Perspectivo)

Sendo assim, para se realizar o efeito contrário (dado um ponto no espaço pós-perspectivo, obter-se o mesmo no espaço global), pode-se derivar a expressão:

$$p_{pos} = (M_{View} \cdot M_{Projection}) * p_{world}$$

$$(M_{View} \cdot M_{Projection})^{-1} p_{pos} = (M_{View} \cdot M_{Projection})^{-1} (M_{View} \cdot M_{Projection}) * p_{world}$$

$$(M_{View} \cdot M_{Projection})^{-1} p_{pos} = I * p_{world}$$

$$p_{world} = (M_{View} \cdot M_{Projection})^{-1} p_{pos}$$

**Equação 11:** Ponto projetado (Mundo)

Sabendo que $[M_{View} \cdot M_{Projection}]$ é inversível.

[2] A continuidade se dá pelo uso da interpolação. Se a função de interpolação for duplamente derivável, $f_{HF}$ será derivável em todo o intervalo de definição.

PUC-Rio - Certificação Digital Nº 0812608/CA

Logo, o processo de funcionamento do *Projected Grid* é similar ao funcionamento de uma textura projetiva.

Na versão original (Johanson, 2004) transforma-se cada ponto da malha do espaço pós-perspectivo para o espaço do mundo, através da matriz $\left[M_{View} \cdot M_{Projection}\right]^{-1}$, usando a *CPU* e, então, se realiza a amostragem da função de altura. Dessa forma, somente pequenas malhas são plausíveis de serem utilizadas sem comprometer o desempenho.

Nesse trabalho, utiliza-se uma adaptação de tal algoritmo para utilizar melhor os recursos presentes nas placas de vídeo modernas. Por conseguinte, o algoritmo passa a ser executado inteiramente na placa de vídeo, no caso, no programa de vértice.

A geração da malha discretizada no espaço pós-perspectivo consiste em definir as coordenadas dos vértices no intervalo de [-1,1] x [-1,1]. Portanto, numa malha N x N cada vértice possui coordenadas $p_{i,j} = \left(\frac{2i}{N} - 1, \frac{2j}{N} - 1, 0\right), i, j \in [0, N-1]$. Na Figura 14 observa-se uma transformação de translação e escala.

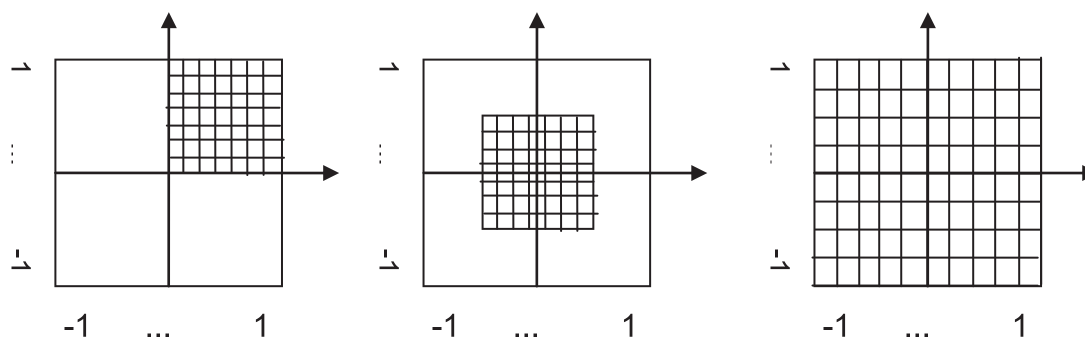



**Figura 14:** Grade no Espaço Pós-Perspectivo

O processo de geração dos dados pode ser desenvolvido seguindo o algoritmo proposto na Tabela 5:

PUC-Rio - Certificação Digital Nº 0812608/CA

**Tabela 5:** Algortimo de Geração do *Projected Grid*

```
01 void ProjectedGridLodManager::SetUpVertices()
02 {
03     VECTOR3* vertices;
04     short* indices;
05     VertexBuffer* mVertexBuffer = NULL;
06     IndexBuffer *mIndexBuffer = NULL;
07
08     // Create the vertex buffer.
09     CreateVertexBuffer( Size * Size * sizeof( VECTOR3), &mVertexBuffer);
10
11        m_pVertexBuffer->Lock( 0, Size * Size * sizeof( VECTOR3 ));
12
13     for (register int x = 0; x < Size; x++)
14     {
15         for (register int y = 0; y < Size; y++)
16         {
17             float i = x / (float)Size;
18             float j = y / (float)Size;
19
20             vertices[x + y * Size].x = 2 * i + 1.0;
21             vertices[x + y * Size].z = 0;
22             vertices[x + y * Size].y = 2 * j + 1.0;
23         }
24     }
25
26     mVertexBuffer->Unlock();
27
28     vertices = NULL;
29
30        CreateIndexBuffer((Size-1)*(Size - 1) * 6 * sizeof( short ),
&mIndexBuffer);
31
32        mIndexBuffer->Lock( 0, 0, (void**)&indices);
33
34     m_BufferSize = (Size - 1) * (Size - 1) * 6;
35
36     int ProjCounter = 0;
37     for (int y = 0; y < Size - 1; y++)
38     {
39         for (int x = 0; x < Size - 1; x++)
40         {
41             int lowerLeft = x + y * Size;
42             int lowerRight = (x + 1) + y * Size;
43             int topLeft = x + (y + 1) * Size;
44             int topRight = (x + 1) + (y + 1) * Size;
45
46             indices[ProjCounter++] = topLeft;
47             indices[ProjCounter++] = lowerRight;
48             indices[ProjCounter++] = lowerLeft;
49
50             indices[ProjCounter++] = topLeft;
51             indices[ProjCounter++] = topRight;
52             indices[ProjCounter++] = lowerRight;
53         }
54     }
55
56     mIndexBuffer->Unlock();
57     indices = NULL;
58 }
```

O algoritmo da Tabela 5 é similar aos anteriores, com uma ligeira modificação nas linhas 17 a 22, sendo essas as alterações para a geração da malha no espaço pós-perspectivo.

PUC-Rio - Certificação Digital Nº 0812608/CA

### 3.3.1. Problemas do Mapeamento Pós-Perspectivo

Contudo, o algoritmo possui os problemas inerentes à transformação perspectiva. Um exemplo clássico pode ser visto na Figura 15.

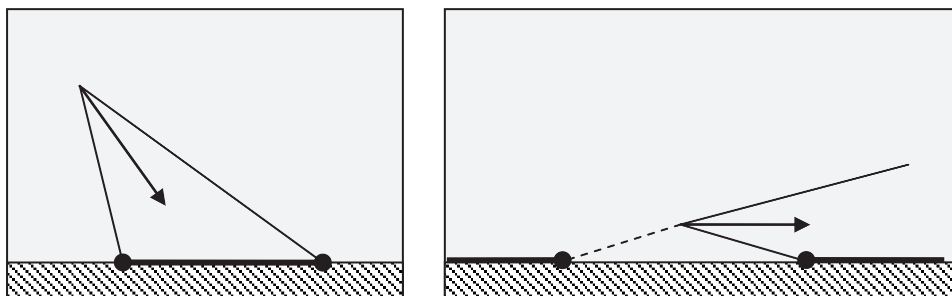


**Figura 15:** Problemas da Projeção

No exemplo, a imagem da esquerda mostra um caso no qual o mapeamento é correto, contudo, na imagem da direita a transformação começa no espaço visível e se estende até o infinito, fazendo com que vértices que deveriam aparecer atrás do observador sejam mapeados para frente do mesmo, causando, assim, um grave problema de visualização. Esse problema também ocorre na técnica de mapas de sombra *Perspective Shadow Maps* (Kozlov, 2004), no entanto, a solução aqui é diferente.

O problema ocorre em situações nas quais a câmera aponta para o horizonte, dessa forma, é necessário que o plano *base* intersecte o volume de visão e o divida em duas partes, sem que haja uma interseção com o *Near* ou *Far plane* pelo plano base. Como há ainda uma alteração na altura dos vértices projetados da malha, isso deve ser garantido para os planos *lower* e *upper* da Figura 13.

PUC-Rio - Certificação Digital Nº 0812608/CA

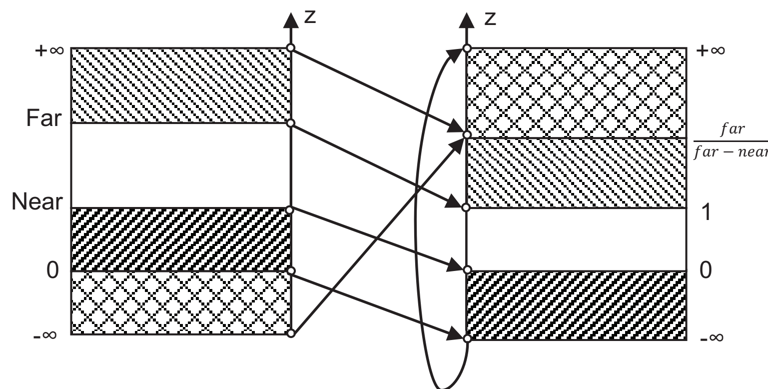



**Figura 16:** Remapeamento da Transformação Perspectiva

Para melhor entender o problema, basta observar a Figura 16, que apresenta o que ocorre com a coordenada z quando a mesma passa por uma transformação perspectiva[3]. Nela, os pontos do intervalo [Near, Far] são levados para [0,1], respectivamente. Contudo, pontos [0,-∞) são mapeados para o intervalo [Far/(Far-Near), +∞), sendo que -∞ é mapeado para Far/(Far-Near) e o zero é mapeado para +∞, analisando $\lim_{z \to 0^-} T_{Perpesctive}(z)$.

Entretanto $\lim_{z \to 0^+} T_{Perpesctive}(z)$ leva o zero para -∞. Outro mapeamento importante é o do intervalo de [Far, +∞) para [1,Far/(Far-Near)]. Por fim, intervalos infinitos são remapeados para intervalos finitos e intervalos infinitos são mapeados para intervalos finitos, além de trazer elementos que antes localizavam-se atrás do observador para a frente desse.

A forma de se resolver o problema consiste em evitar as seguintes situações com a câmera, sabendo que, o projetor possui a mesma matriz de projeção que a câmera do observador:

- O projetor não deve olhar para muito distante do plano base (visão do horizonte).

[3] A transformação utilizada para a coordenada z é $z' = \frac{z - near}{far - near} \cdot \frac{1}{z}$, ou seja, não mapea para o intervalo [-1,1].

PUC-Rio - Certificação Digital Nº 0812608/CA

- A posição do projetor deve ficar fora do volume de malha visível.[4]

Com esses cuidados os problemas de projeção são resolvidos, contudo, isso limitaria os movimentos possíveis do usuário. Dessa forma, cria-se uma câmera especial na cena, a câmera de projeção da malha. Essa câmera que se encarregará de respeitar as regras de posicionamento.

O projetor possui a mesma posição da câmera, no entanto, ela pode ser mudada, dependendo das restrições acima. Pelas restrições, a posição do projetor deve sempre ficar acima de $S_{upper}$ ou abaixo de $S_{lower}$, lembrando que o sentido renderização (horário e anti-horário) da malha é diferente se o observador estiver embaixo da malha ou em cima da mesma. Isso já garante que o projetor não pertencerá ao $V_{visible}$, pois o mesmo não está no volume de variação da malha ($V_{displaceable}$), dessa forma: $P_{projector} \notin V_{displaceable}$, como $V_{visible} = V_{displaceable} \cap V_{Cam}$, $P_{projector} \notin V_{visible}$.

Outro cuidado a ser tomado com o vetor de visão do projetor, é que o mesmo não deve ser paralelo ao plano $S_{base}$. Johanson (2004) apresenta em seu trabalho duas formas de se computar o vetor de visão (*look-at*). No primeiro método, o vetor de visão é calculado como sendo o vetor unitário (versor) que aponta da posição do projetor até o ponto de interseção da reta, que é formada pela posição do observador e o vetor de visão do mesmo, com o plano $S_{base}$ (Figura 17.a). O outro método calcula um ponto a uma distância fixa do observador ($P = P_{cam} + V_{view} \cdot d_{fixed}$) e projeta-o sobre o plano $S_{base}$ (Figura 17.b).

[4] O volume de malha visível ($\mathbf{V_{visible}}$) é a interseção de $\mathbf{V_{displaceable}}$ com o *Frustum* da câmera ($\mathbf{V_{Cam}}$). Logo: $V_{visible} = V_{displaceable} \cap V_{Cam}$

PUC-Rio - Certificação Digital Nº 0812608/CA

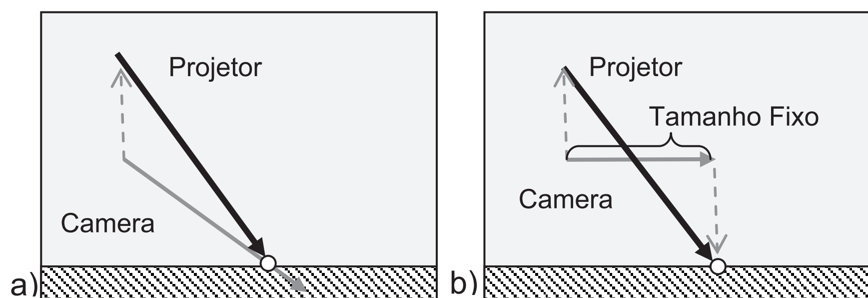



**Figura 17:** Métodos de Cálculo do Vetor de Visão do Projetor

Dessa forma, o primeiro método é adequado à utilização quando o observador está olhando em direção ao plano do oceano e, o segundo, quando o mesmo está olhando para o horizonte.

### 3.3.2. Reescalamento do plano pós-perspectivo

Devido à utilização de uma câmera adicional para a projeção dos vértices pós-perspectivos, não há mais garantia de que todos os vértices aparecerão na imagem final, pois, agora há dois volumes de visão. Uma estratégia para aumentar o número de vértices na imagem final é utilizar uma matriz de conversão de extensão ($M_{range}$). O conceito dessa transformação é reposicionar os vértices projetados pelo projetor dentro da região de visualização da câmera do observador. A criação dessa matriz é realizada por meio dos seguintes passos.

1. Transformam-se os pontos de fronteira $(\pm1, \pm1, \pm1)$ do *frustum* do observador para o espaço do mundo utilizando a matriz inversa de visão e projeção ($M_{viewproj}$) da câmera do observador.

2. Calculam-se as interseções das arestas do *frustum* transformado com os planos de fronteira do oceano ($S_{base}$, $S_{upper}$, $S_{lower}$). Armazena-se esses pontos em um *buffer* (Figura 18.a).

PUC-Rio - Certificação Digital Nº 0812608/CA

3. Qualquer ponto de fronteira tranformado do *frustum*, que estiver contido entre os planos de fronteira do oceano deve ser adicionado ao *buffer* também.

4. Se não existir nenhum ponto no *buffer*, isto significa que não há interseção entre $V_{displaceable}$ e $V_{view}$, logo $V_{visible}$ é vazio e o oceano não deve ser desenhado. Nesse caso, o processamento é encerrado.

5. Projeta-se todos os pontos do buffer sobre o plano $S_{base}$ (Figura 18.b).

6. Transforma-se todos os pontos projetados para o espaço do projetor (pós-perspectivo) usando a matriz de visão e projeção ($M_{viewproj}$) do projetor (Figura 18.c,d). Observa-se que é utilizada a matriz de visão do projetor e não a da câmera do observador.

7. Constrói-se a matriz $M_{range}$ como na Equação 12, onde os máximos e mínimos são definidos como os máximos e mínimos dos pontos projetados. Observa-se que as coordenadas z e w são mantidas.

$$M_{range} = \begin{bmatrix} x_{max} - x_{min} & 0 & 0 & x_{min} \\ 0 & y_{max} - y_{min} & 0 & y_{min} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

**Equação 12:** Matriz $M_{range}$

PUC-Rio - Certificação Digital Nº 0812608/CA

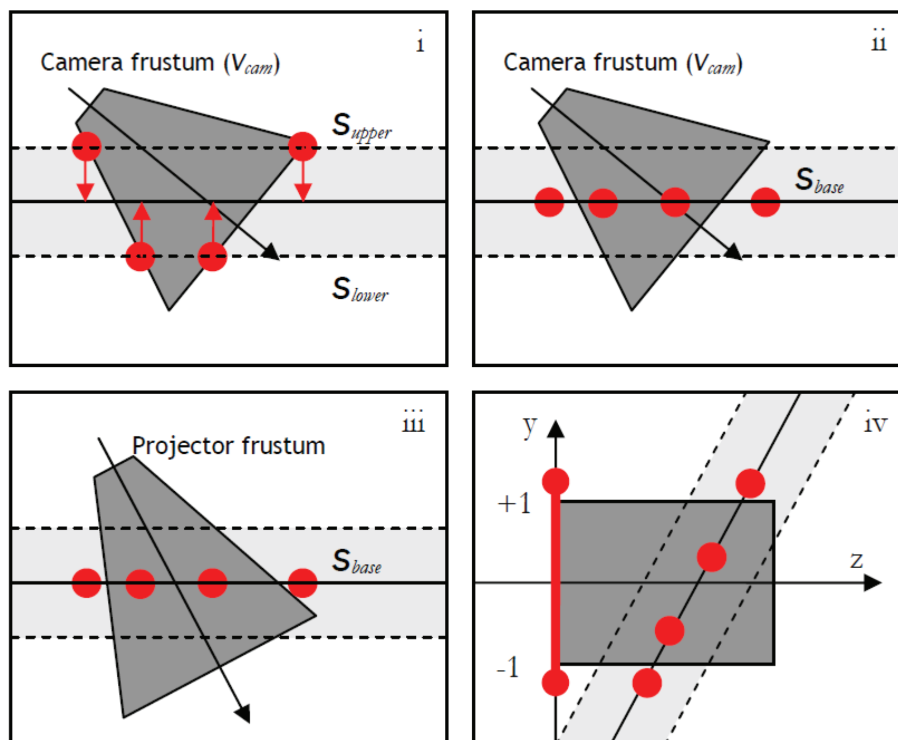



**Figura 18:** Cálculo da Matriz $M_{range}$ (Johanson, 2004)

### 3.3.3. Algoritmo

Resolvendo-se os problemas gerados pela projeção perspectiva, pode-se utilizar os seguintes passos para a utilização do algoritmo.

1. Obtem-se a posição, direção e todos os parâmetros necessários para a replicação das matrizes da câmera do observador.

2. Inicia-se o processo de criação da matriz $M_{range}$ (conforme explicado em 3.3.2). Caso nenhuma parte de $V_{displaceable}$ intercepte o volume de visão da câmera do observador, aborte a renderização.

3. Utilizando as regras descritas em 3.3.1, cria-se a matriz de visão do projetor ($M_{pview}$) e utilizando a matriz de projeção da câmera do observador ($M_{perspective}$) cria-se a matriz do projetor ($M_{projector}$), dada por: $M_{projector} = [M_{pview} \cdot M_{perspective}]^{-1}$

PUC-Rio - Certificação Digital Nº 0812608/CA

4. Finaliza-se o cálculo da matriz $M_{range}$ e atualiza-se a matriz $M_{projector}$ como:

$$M_{projector} = M_{range} \cdot [M_{pview} \cdot M_{perspective}]^{-1}$$

**Equação 13:** Matriz do projetor

5. Utilizando a placa gráfica no programa de vértice, cada vértice da malha é transformado pela matriz $M_{projector}$ duas vezes, uma com a coordenada z em -1($P_a$) e uma como +1 ($P_b$). O vértice final será a interseção entre a linha formada pelos vértices $P_a$ e $P_b$ e o plano $S_{base}$.

6. Utilizando um mapa de altura, acessa-se um texel para obter a amplitude do vértice desejado.

7. Prossegue-se com os cálculos habituais de projeção da câmera do observador.

A Figura 19 mostra o resultado da técnica. Dentre as técnicas estudadas, essa é a mais elegante do ponto de vista da computação gráfica, pois, explora os conceitos mais essenciais da mesma.

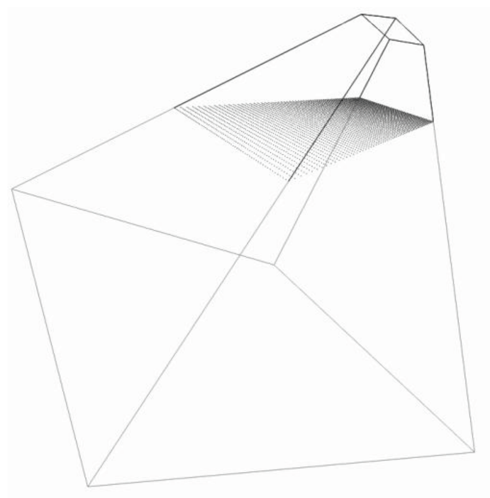


**Figura 19:** Resultado da Técnica de Projected Grid (Johanson, 2004)

PUC-Rio - Certificação Digital Nº 0812608/CA

Terminado a análise das técnicas de nível de detalhe, pode-se iniciar a análise das técnicas de simulação da malha do oceano, isentando-se da forma de representação em larga escala da mesma.

PUC-Rio - Certificação Digital Nº 0812608/CA