6
Mapeamento de Propriedades

Um grande desafio no uso de estruturas de multi-resolução para a visualização científica de modelos massivos é o tratamento das propriedades associadas à malha original do modelo. No caso de modelos de reservatórios, cada simulação conterá dezenas de campos escalares e vetoriais associados a cada célula da malha, e cada campo tipicamente varia ao longo de dezenas de passos de simulação (*time steps*).

Um primeiro tratamento que rapidamente se prova inviável seria obter as propriedades, que são definidas nas células do modelo original, e calcular uma suavização para estas propriedades nos vértices dessa malha. Em seguida, incluiríamos cada propriedade suavizada como atributo de vértice para fins de simplificação. A simplificação trataria de interpolar esses atributos ao criar ou mover vértices na malha. Além do enorme volume de dados e da grande quantidade de interpolações lineares feitas, a estrutura de multi-resolução gerada seria utilizável para a visualização dos resultados de uma única simulação, forçando uma nova geração dessa estrutura a cada nova simulação numérica. Além disso, conforme a simplificação avança, as variações dadas pelos gradientes dos campos escalares e vetoriais associados ao reservatório vão sendo perdidas por conta da aproximação linear feita a cada interpolação.

A utilização de representações simplificadas do modelo apresenta mais um desafio na renderização de informações associadas à topologia original do modelo: a visualização do *wireframe* da malha original, que pode ser útil para revelar informações estruturais sobre a malha utilizada para modelar o domínio do reservatório.

O trabalho descrito neste capítulo propõe lidar com o problema de visualização de propriedades associadas à malha original do reservatório sobre as malhas simplificadas pelo uso de mapeamento de texturas. Ele introduz o uso de uma textura 3D, aqui denominada *textura de propriedade*, para a visualização das propriedades associadas às células do modelo original. Essa textura 3D possui as mesmas dimensões da grade topológica do reservatório, contendo $ni \times nj \times nk$ *voxels*. A proposta de mapeamento de propriedade inclui a visualização da propriedade não suavizada (por célula) e a visualização da propriedade suavizada nos vértices da malha. É feita também uma proposta para a visualização do *grid* original mapeado sobre as malhas simplificadas utilizadas, também fazendo o uso de texturas. Ambas as propostas são compostas pelo cálculo apropriado de coordenadas de textura associadas aos vértices

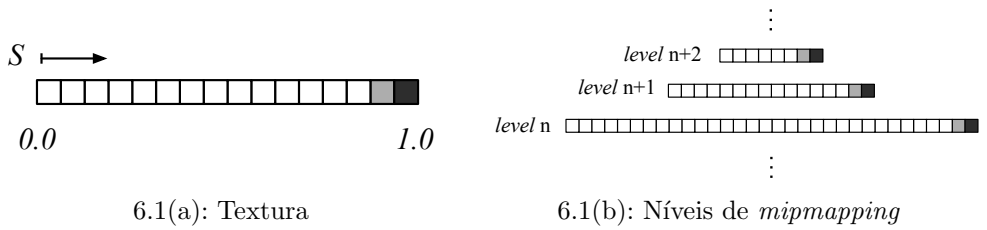
da malha original, que serão interpoladas enquanto a simplificação avança. Como essas coordenadas são independentes dos valores das propriedades, esta técnica dispensa a inclusão de valores de propriedades no cálculo da hierarquia de multi-resolução, o que permite que ela seja reutilizada em diferentes simulações do mesmo modelo.

As imagens obtidas ao se empregar as técnicas propostas sobre as malhas simplificadas da multi-resolução foram praticamente idênticas às imagens obtidas pela renderização do modelo original, havendo um pequeno impacto no desempenho da visualização quando comparado com a visualização sem o mapeamento desses dados.

Em primeiro lugar apresentaremos a nossa proposta para o mapeamento do *grid* original sobre a malha simplificada. Em seguida descreveremos o mapeamento das propriedades sobre as malhas com multi-resolução. Por fim, será feita uma breve análise do impacto desses mapeamentos sobre o desempenho da visualização.

6.1 Mapeamento do Grid Original

Em (14, 15), apresentamos a textura de *wireframe*, que é uma técnica simples e eficiente para a renderização do *wireframe* de malhas com base no mapeamento de texturas. O algoritmo renderiza o *wireframe* em uma única passada de renderização, consistindo basicamente no uso do mapeamento de uma textura para desenhar as arestas da malha em conjunto com a rasterização dos seus polígonos. Esse mapeamento de texturas é feito utilizando uma textura RGBA 1D, a *textura de wireframe*, que representa a metade da linha (ao longo da direção da sua espessura). Cada polígono desenha a metade da linha, e a renderização da linha se completa após dois polígonos adjacentes serem rasterizados. A Figura 6.1(a) ilustra a textura utilizada para renderizar linhas de *wireframe* com um valor de espessura igual a 3,0. O canal alfa, que é ilustrado na figura, codifica a opacidade. Neste caso, a metade da espessura



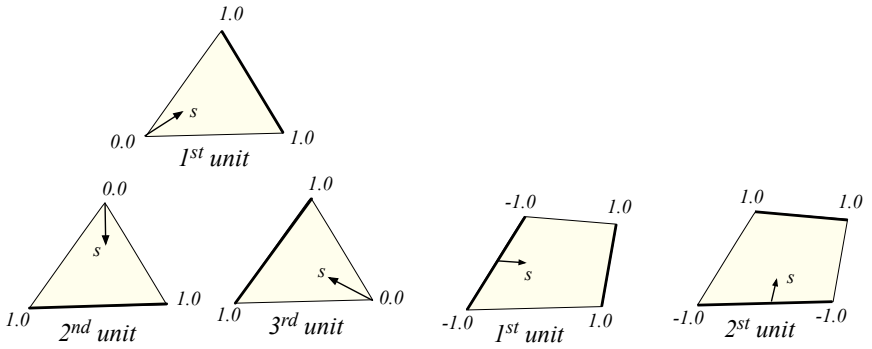
6.1(a): Textura 6.1(b): Níveis de *mipmapping*

Figura 6.1: Textura utilizada para renderizar o *wireframe* no trabalho (14). A imagem ilustra os valores de opacidade atribuídos aos *texels*.

da linha (um valor de 1,5) é representado ao se atribuir um valor de alfa igual a 1,0 para o último *texel* e igual a 0,5 para o seu *texel* vizinho. Habilita-se o uso de *mipmapping* para garantir a espessura da linha independente do tamanho da primitiva quando mapeada para a tela. Os *texels* representando o *wireframe* são preservados em cada nível da pirâmide de *mipmapping*, ilustrada na Figura 6.1(b).

A obtenção dos resultados visuais desejados é feita pela atribuição de coordenadas de texturas apropriadas ao se desenhar as primitivas gráficas. No caso de triângulos eles utilizam três unidades de textura com o mesmo objeto de textura carregado. Cada aresta do triângulo é renderizada por uma dessas unidades de textura: cada unidade terá uma coordenada de textura igual a 0,0 para um vértice e igual a 1,0 para os outros dois vértices, conforme ilustrado na Figura 6.2(a). Para um quadrilátero basta utilizar duas unidades de textura, atribuindo as coordenadas de texturas -1,0 e 1,0 a vértices em lados opostos e mapeando a textura no modo de *wrapping mirrored-repeat* (Figura 6.2(b)). Os valores RGB dos *texels* são utilizados para codificar a cor da linha de *wireframe*, e a função de textura é configurada para o modo *decal*. Esse trabalho também apresenta uma forma para a renderização do *wireframe* isoladamente. Além de permitir a renderização das linhas do *wireframe* em conjunto com a superfície em apenas uma passada, o que é importante em termos de eficiência, a técnica possui um tratamento automático de serrilhamento (*aliasing*) pelo seu uso de mapeamento de texturas.

O mesmo trabalho também apresenta um algoritmo para a atribuição de coordenadas de textura para malhas de polígonos, onde é importante o reuso de vértices para a obtenção de uma visualização eficiente. No caso de uma malha estruturada de quadriláteros, é bem simples atribuir coordenadas



6.2(a): Triângulo 6.2(b): Quadrilátero

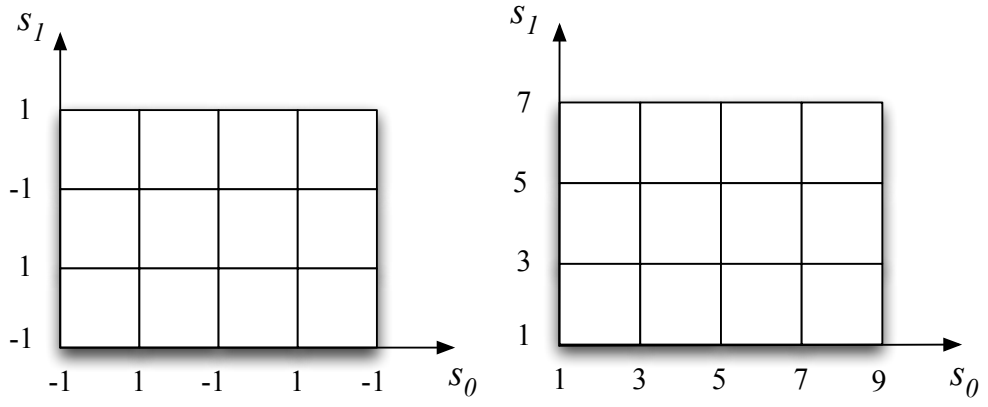
Figura 6.2: Coordenadas de textura atribuídas aos vértices para a renderização das arestas de triângulos e quadriláteros.

de textura não conflitantes para a renderização do *wireframe*: basta alternar as coordenadas de textura -1,0 e 1,0 nas duas direções, onde cada direção corresponde a uma unidade de textura, conforme ilustrado na Figura 6.3(a).

É possível também especificar as coordenadas de textura para esse tipo de malha com base em um sistema global de coordenadas: basta atribuir coordenadas de texturas às linhas e colunas de arestas como valores ímpares e consecutivos, conforme ilustrado na Figura 6.3(b). O uso de um sistema global de coordenadas torna possível renderizar o *wireframe* da malha independente da malha utilizada para representar o modelo, o que tem aplicações importantes, especialmente na área de visualização com multi-resolução.

Consideremos uma malha de reservatórios 2D, onde cada célula é representada por um quadrilátero com índices i e j em uma grade topológica $ni \times nj$. Torna-se bem simples atribuir coordenadas de textura aos vértices dessa malha: assumindo que os índices começam em 0, as seguintes coordenadas globais de textura são atribuídas aos vértices de uma célula de índices $[i, j]$: $[2i+1, 2j+1]$, $[2(i+1)+1, 2j+1]$, $[2(i+1)+1, 2(j+1)+1]$ e $[2i+1, 2(j+1)+1]$. Denomina-se neste trabalho as coordenadas i, j que originam essas coordenadas globais de textura as *coordenadas topológicas* de cada vértice. É fácil verificar que essa forma de atribuir coordenadas funciona bem mesmo na presença de falhas geológicas – haverá no espaço topológico vértices diferentes na mesma posição no sistema global de coordenadas, o que não é um problema para a rasterização de suas primitivas.

A extensão dessa técnica de atribuição de coordenadas de textura para a nossa malha semi-estruturada de hexaedros é feita pela adição da coordenada topológica k. No entanto, como o objetivo é desenhar a grade topológica sobre



6.3(a): Sistema local 6.3(b): Sistema global

Figura 6.3: Atribuição de coordenadas de textura para malhas de quadriláteros topologicamente estruturadas.

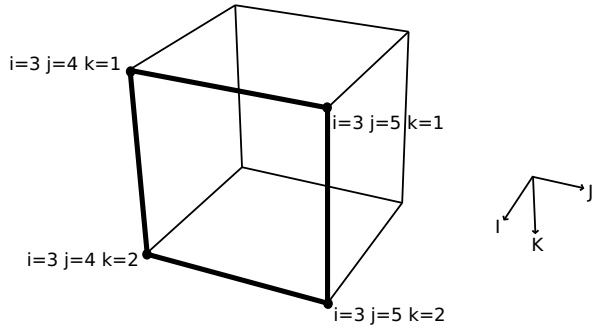


Figura 6.4: Exemplo do mapeamento de *grid* em 3D: as coordenadas i são constantes ao longo da face.

as faces externas da malha de reservatórios, é necessário tomar um cuidado adicional na passagem da coordenada topológica i, j, k para o mapeamento da textura de *wireframe*. Consideremos a face destacada na Figura 6.4. Caso utilizemos o esquema de atribuição de coordenadas descrito acima diretamente em três indexações da textura de *wireframe*, todos os fragmentos gerados na rasterização da face ressaltada no exemplo receberão erradamente a cor configurada para o desenho do grid, pois a coordenada i possui valor inteiro e constante ao longo de toda a face. O problema em 3D passa a ser a identificação de duas coordenadas para o mapeamento da textura de *wireframe* dentre as três coordenadas topológicas de cada vértice. A proposta deste trabalho é empregar um *geometry shader* para identificar e excluir a coordenada topológica que menos varia ao longo de cada primitiva. Essa identificação é feita pelo cálculo de uma espécie de "vetor normal topológico" da primitiva, calculado pelo produto vetorial no espaço de coordenadas topológicas. Essa normal terá maior valor absoluto de coordenada no eixo em que há a menor variação de coordenada topológica. Uma vez detectada qual coordenada varia menos, a sua exclusão é feita pela simples atribuição da coordenada em questão para o valor de 0,0, que forçará o acesso a um *texel* da textura não pintado com a cor do *wireframe* na unidade de textura em questão. A Listagem 6.1 mostra o código fonte do *geometry shader* empregado, que manipula as coordenadas topológicas conforme aqui descrito e as armazena nas coordenadas de textura de índice 5, que serão repassadas para o acesso à textura de *wireframe* na nossa implementação.

Como cada vértice da malha está posicionado em um sistema global de coordenadas topológicas, é possível incluir essas coordenadas como atributos de vértice no nosso simplificador de malhas de hexaedros. Em primeiro lugar atribuímos as coordenadas topológicas a cada vértice da malha original segundo o sistema global de coordenadas de textura descrito acima estendido

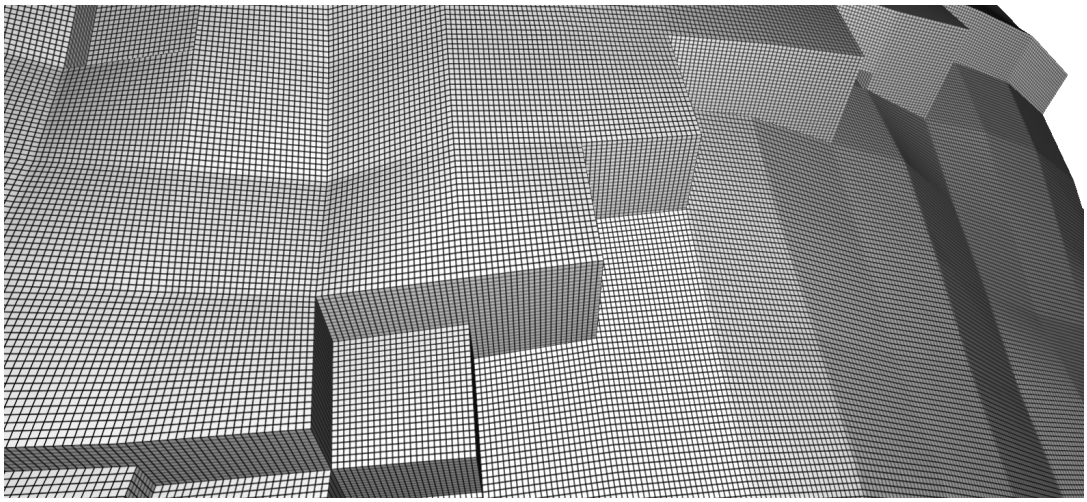
para considerar a coordenada k das células. A partir disso, cada colapso de coluna interpolará linearmente esses atributos sempre que um vértice for criado ou reposicionado. A mesma interpolação será feita no processo de simplificação das faces laterais externas descrito no capítulo anterior. Este procedimento nos permitirá renderizar a grade original do modelo desacoplada da malha de hexaedros utilizada, funcionando bem em praticamente qualquer nível de simplificação da estrutura de multi-resolução.

A Figura 6.5 ilustra os resultados visuais obtidos: a Figura 6.5(a) mostra a malha original do reservatório com o desenho do *grid* correspondente; a Figura 6.5(b) mostra uma malha simplificada; e a Figura 6.5(c) mostra o *grid* original mapeado sobre a malha simplificada. Conforme pode ser notado, a imagem obtida é praticamente idêntica à imagem gerada com o modelo original em termos do desenho do *grid* topológico original, podendo haver apenas algumas pequenas distorções no mapeamento.

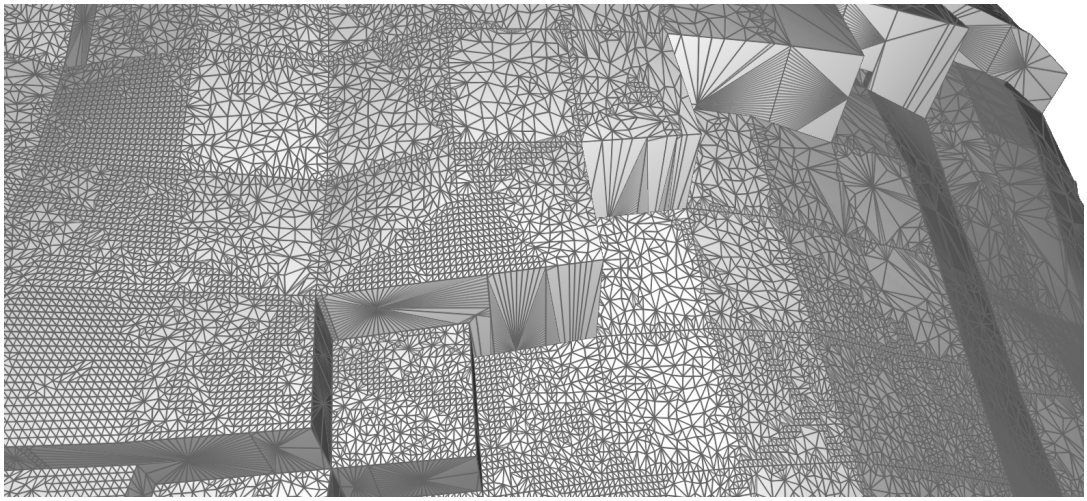
Listagem 6.1: Código fonte do *geometry shader* empregado para a manipulação das coordenadas de textura de cada polígono no mapeamento do *grid* original.

```
#version 120
#extension GL_EXT_geometry_shader4 : enable
#extension GL_EXT_gpu_shader4 : enable

void main ()
{
  vec3 v01 = normalize(gl_TexCoordIn[1][1].stp - gl_TexCoordIn[0][1].stp);
  vec3 v12 = normalize(gl_TexCoordIn[2][1].stp - gl_TexCoordIn[1][1].stp);
  vec3 top_normal = cross(v01, v12);
  vec3 top_normal_abs = vec3(abs(top_normal.s),
                             abs(top_normal.t),
                             abs(top_normal.p));

  vec4 grid_tex_coord_0 = 2*gl_TexCoordIn[0][1] + vec4(1.0, 1.0, 1.0, 0.0);
  vec4 grid_tex_coord_1 = 2*gl_TexCoordIn[1][1] + vec4(1.0, 1.0, 1.0, 0.0);
  vec4 grid_tex_coord_2 = 2*gl_TexCoordIn[2][1] + vec4(1.0, 1.0, 1.0, 0.0);

  if (top_normal_abs.s > top_normal_abs.t) {
    if (top_normal_abs.s > top_normal_abs.p) {
      // 'i' has smallest variation: exclude coordinate from grid rendering
      // by setting to value 0.0
      grid_tex_coord_0.s = 0.0;
      grid_tex_coord_1.s = 0.0;
      grid_tex_coord_2.s = 0.0;
    }
    else {
      // 'k' has smallest variation: exclude coordinate from grid rendering
      // by setting to value 0.0
      grid_tex_coord_0.p = 0.0;
      grid_tex_coord_1.p = 0.0;
      grid_tex_coord_2.p = 0.0;
    }
  }
  else {
    if (top_normal_abs.t > top_normal_abs.p) {
```

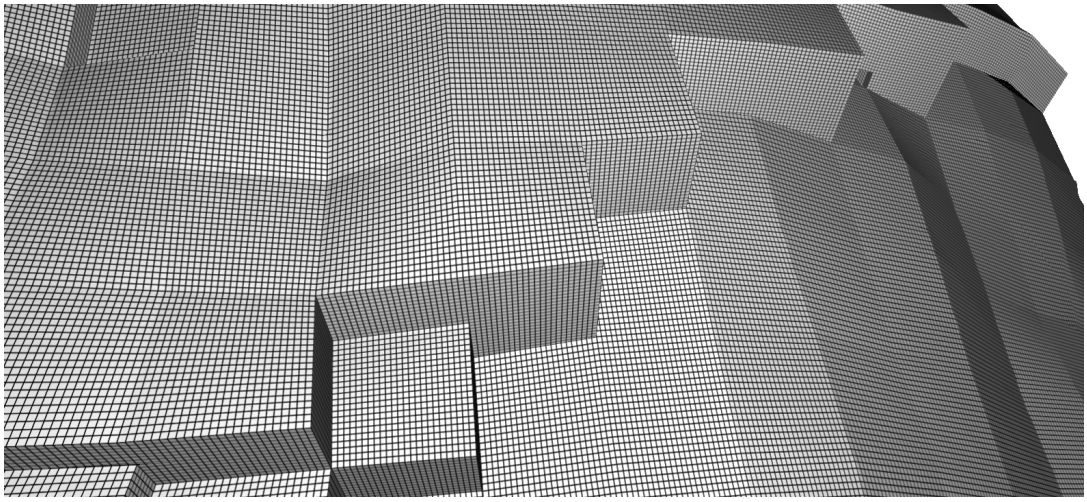
```
36        // 'j' has smallest variation: exclude coordinate from grid rendering
37        // by setting to value 0.0
38        grid_tex_coord_0.t = 0.0;
39        grid_tex_coord_1.t = 0.0;
40        grid_tex_coord_2.t = 0.0;
41      }
42      else {
43        // 'k' has smallest variation: exclude coordinate from grid rendering
44        // by setting to value 0.0
45        grid_tex_coord_0.p = 0.0;
46        grid_tex_coord_1.p = 0.0;
47        grid_tex_coord_2.p = 0.0;
48      }
49    }
50    gl_Position    = gl_PositionIn[0];
51    gl_FrontColor  = gl_FrontColorIn[0];
52    gl_TexCoord[5] = grid_tex_coord_0; // grid texture coordinates
53    EmitVertex();
54    gl_Position    = gl_PositionIn[1];
55    gl_FrontColor  = gl_FrontColorIn[1];
56    gl_TexCoord[5] = grid_tex_coord_1; // grid texture coordinates
57    EmitVertex();
58    gl_Position    = gl_PositionIn[2];
59    gl_FrontColor  = gl_FrontColorIn[2];
60    gl_TexCoord[5] = grid_tex_coord_2; // grid texture coordinates
61    EmitVertex();
62    EndPrimitive();
63  }
```



6.5(a): Malha original



6.5(b): Malha simplificada



6.5(c): *Grid* original mapeado na malha simplificada

Figura 6.5: Renderização do *grid* original desacoplado do suporte geométrico utilizado.

6.2
Mapeamento de Propriedades

Assim como foi feito para o mapeamento do *grid* original, este trabalho propõe lidar com o problema de visualização de propriedades associadas à malha original do reservatório pelo uso de mapeamento de texturas e pelo cálculo apropriado das coordenadas de textura para cada vértice dessa malha.

Em tempo de visualização especificaremos uma textura 3D, denominada *textura de propriedade*, sempre que um *time step* de uma propriedade for requisitado. Essa textura 3D possui as mesmas dimensões e esquema de indexação da grade topológica do reservatório, contendo $ni \times nj \times nk$ *voxels*, onde cada *voxel* de índice i, j, k armazena o valor de propriedade associado à célula de mesmos índices no modelo original. Caso a propriedade requisitada seja escalar, a textura de propriedade possuirá 1 valor de ponto flutuante por *voxel*. Caso a propriedade seja vetorial, cada *voxel* da textura possuirá 3 valores de ponto flutuante. Uma propriedade pode ser definida para todas as células ou apenas para as células ativas. Caso a propriedade seja definida para todas as células, basta associar o valor de cada célula ao *voxel* correspondente e o cálculo da textura está completo. Caso a propriedade não seja definida para células inativas, os *voxels* associados às células inativas usadas para completar as colunas no procedimento de multi-resolução têm seus valores dados pelas médias dos *voxels* vizinhos associados a células ativas.

Primeiro, apresentaremos o método proposto para o mapeamento dessa textura de forma a permitir a visualização da propriedade por célula do modelo original (propriedade não suavizada). Em seguida, é apresentada uma proposta de mapeamento para permitir a visualização dessas propriedades suavizadas nos vértices da malha.

Assim como no caso do *grid* original, os métodos propostos são facilmente integrados com a visualização com multi-resolução, o que é feito pela inclusão de suas coordenadas de textura como atributos dos vértice da malha original no procedimento de geração da estrutura de multi-resolução.

6.2.1
Mapeamento da Propriedade Não Suavizada

O problema de mapeamento da propriedade não suavizada, onde há um valor por célula do modelo original, é ligado ao de mapeamento do *grid* original. Deseja-se que cada fragmento gerado pela rasterização de uma face externa seja pintado com a cor associada ao valor de propriedade da célula onde o fragmento se encontraria no modelo original. Assim como no mapeamento do *grid* original, podemos utilizar as coordenadas topológicas i, j, k associadas a

cada vértice para indexar a textura de propriedade. Isso é feito desabilitando o uso de filtros no mapeamento da textura (parâmetro `GL_NEAREST` na API do OpenGL).

A Figura 6.4 da Seção 6.1 ilustrou o que ocorre ao longo de uma face no que tange ao acesso à textura de *wireframe*. Caso utilizemos a coordenada i, j, k diretamente para indexar a textura de propriedade, é possível que ocorram problemas de precisão numérica ao longo das faces, pois o mapeamento `GL_NEAREST` poderá ora acessar o *voxel* associado à célula desejada, ora acessar o *voxel* associado à outra célula adjacente a esta face.

Nossa proposta lida com esse problema de precisão empregando um *geometry shader* similar ao descrito na Seção 6.1. O objetivo é novamente manipular as coordenadas i, j, k associadas aos vértices de cada primitiva. O *geometry shader* detectará, assim como no caso do *grid* original, a coordenada topológica que varia menos ao longo de cada face, o que será feito novamente pelo uso de um vetor normal topológico. Caso essa coordenada seja constante ao longo de todo o polígono e possua um valor inteiro, aplicaremos um deslocamento no seu valor com o intuito de posicionar a coordenada no centro da célula a ser indexada, evitando assim o problema de precisão aqui descrito. O deslocamento será de -0,5 caso a normal esteja voltada na direção positiva do eixo associado à coordenada em questão e de +0,5 caso contrário, o que é decidido pelo sinal da componente da normal nesse eixo. A Listagem 6.2 mostra o código fonte do *geometry shader* empregado, que desloca as coordenadas topológicas caso necessário e as armazena nas coordenadas de textura de índice 1, que serão repassadas para o acesso à textura 3D na nossa implementação.

As Figuras 6.6 e 6.7 ilustram os resultados visuais obtidos no mapeamento da propriedade não suavizada sobre uma malha simplificada pelo processo de geração da hierarquia de multi-resolução: a Figura 6.6(a) mostra a propriedade mapeada sobre a malha do modelo original; a Figura 6.6(b) mostra a malha simplificada utilizada nas figuras a seguir; a Figura 6.7(a) mostra a propriedade mapeada sobre a malha simplificada; e a Figura 6.7(b) mostra a propriedade e o *grid* original mapeados em conjunto sobre a malha simplificada. Conforme pode ser observado, a imagem obtida pelo mapeamento da propriedade associada às células do modelo original sobre a malha simplificada (Figura 6.7(b)) é praticamente idêntica à imagem obtida utilizando a malha original do reservatório (Figura 6.6(a)).

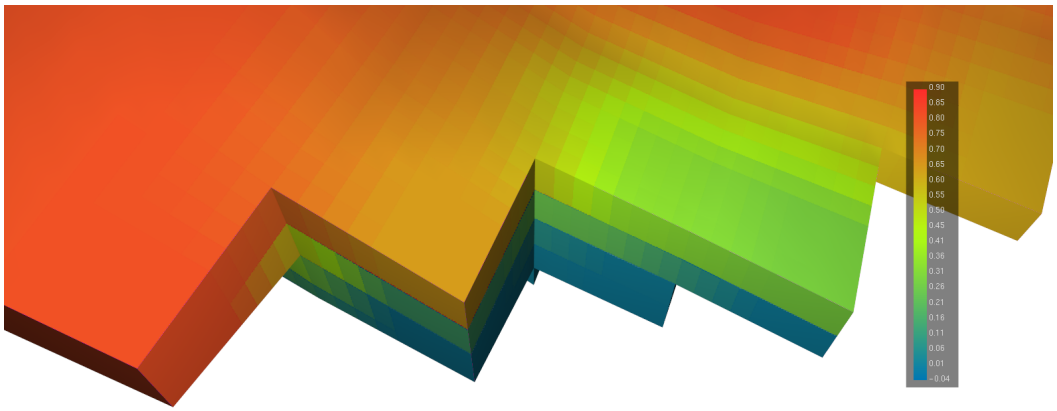
A técnica de deslocamento das coordenadas de textura resolve corretamente os problemas de precisão ao longo de todo o interior das faces das células do modelo original. Esta estratégia possui no entanto uma limitação dada por um problema de precisão nos fragmentos próximos às arestas do modelo origi-

nal, onde ainda pode haver dúvida sobre qual é a célula associada no modelo original. Esse problema pode ser minimizado pelo mapeamento do *grid* original sobre a malha, o que pode ser observado nas figuras. A sua solução definitiva permanece como trabalho futuro.

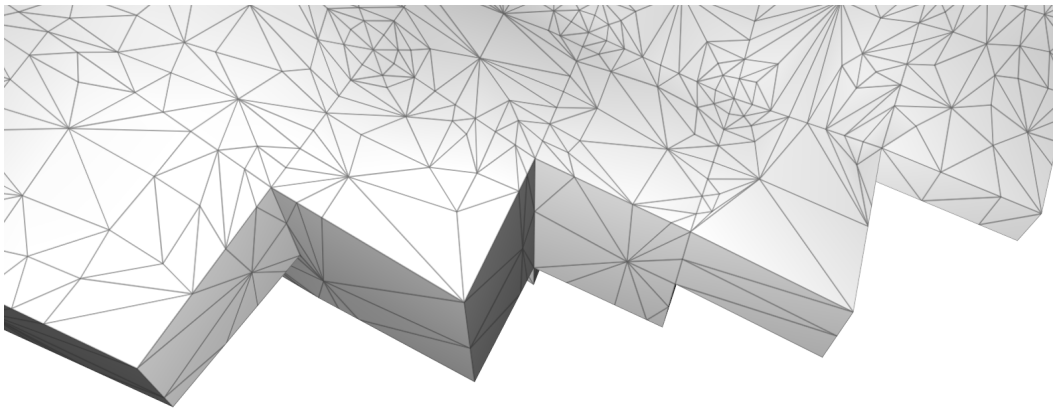
Listagem 6.2: Código fonte do *geometry shader* empregado para a manipulação das coordenadas de textura de cada polígono no mapeamento da propriedade por célula.

```
#version 120
#extension GL_EXT_geometry_shader4 : enable
#extension GL_EXT_gpu_shader4 : enable

bool CloseToInteger (float f, out float fi)
{
  fi = round(f);
  return abs(f - fi) < 0.03;
}

bool CoordConstantAndInteger (float s1, float s2, float s3)
{
  float si1, si2, si3;
  return CloseToInteger(s1, si1) &&
         CloseToInteger(s2, si2) &&
         CloseToInteger(s3, si3) &&
         si1 == si2              &&
         si1 == si3;
}

void main ()
{
  vec3 v01 = normalize(gl_TexCoordIn[1][1].stp - gl_TexCoordIn[0][1].stp);
  vec3 v12 = normalize(gl_TexCoordIn[2][1].stp - gl_TexCoordIn[1][1].stp);
  vec3 top_normal = cross(v01, v12);
  vec4 prop_coord_offsets = vec4(0.0, 0.0, 0.0, 0.0);
  vec3 top_normal_abs = vec3(abs(top_normal.s),
                             abs(top_normal.t),
                             abs(top_normal.p));

  if (top_normal_abs.s > top_normal_abs.t) {
    if (top_normal_abs.s > top_normal_abs.p) {
      // 'i' has smallest variation: see if all vertices 'i' are close to
      // integers and equal. if so, manipulate 'i' coordinate to center of cell
      // (will index the property texture correctly)
      prop_coord_offsets.s = CoordConstantAndInteger(gl_TexCoordIn[0][1].s,
                                                     gl_TexCoordIn[1][1].s,
                                                     gl_TexCoordIn[2][1].s)
                             ?
                               (top_normal.s < 0.0 ? 0.5 : -0.5) :
                               0.0;
    }
    else {
      // 'k' has smallest variation: see if all vertices 'k' are close to
      // integers and equal. if so, manipulate 'k' coordinate to center of cell
      // (will index the property texture correctly)
      prop_coord_offsets.p = CoordConstantAndInteger(gl_TexCoordIn[0][1].p,
                                                     gl_TexCoordIn[1][1].p,
                                                     gl_TexCoordIn[2][1].p)
```

```
50                              ?
51                                (top_normal.p < 0.0 ? 0.5 : -0.5) :
52                                0.0;
53      }
54    }
55    else {
56      if (top_normal_abs.t > top_normal_abs.p) {
57        // 'j' has smallest variation: see if all vertices 'j' are close to
58        // integers and equal. if so, manipulate 'j' coordinate to center of cell
59        // (will index the property texture correctly)
60        prop_coord_offsets.t = CoordConstantAndInteger(gl_TexCoordIn[0][1].t,
61                                                       gl_TexCoordIn[1][1].t,
62                                                       gl_TexCoordIn[2][1].t)
63                              ?
64                                (top_normal.t < 0.0 ? 0.5 : -0.5) :
65                                0.0;
66      }
67      else {
68        // 'k' has smallest variation: see if all vertices 'k' are close to
69        // integers and equal. if so, manipulate 'k' coordinate to center of cell
70        // (will index the property texture correctly)
71        prop_coord_offsets.p = CoordConstantAndInteger(gl_TexCoordIn[0][1].p,
72                                                       gl_TexCoordIn[1][1].p,
73                                                       gl_TexCoordIn[2][1].p)
74                              ?
75                                (top_normal.p < 0.0 ? 0.5 : -0.5) :
76                                0.0;
77      }
78    }
79    gl_Position    = gl_PositionIn[0];
80    gl_FrontColor  = gl_FrontColorIn[0];
81    // per cell property coordinates
82    gl_TexCoord[1] = gl_TexCoordIn[0][1] + prop_coord_offsets;
83    EmitVertex();
84    gl_Position    = gl_PositionIn[1];
85    gl_FrontColor  = gl_FrontColorIn[1];
86    // per cell property coordinates
87    gl_TexCoord[1] = gl_TexCoordIn[1][1] + prop_coord_offsets;
88    EmitVertex();
89    gl_Position    = gl_PositionIn[2];
90    gl_FrontColor  = gl_FrontColorIn[2];
91    // per cell property coordinates
92    gl_TexCoord[1] = gl_TexCoordIn[2][1] + prop_coord_offsets;
93    EmitVertex();
94    EndPrimitive();
95  }
```

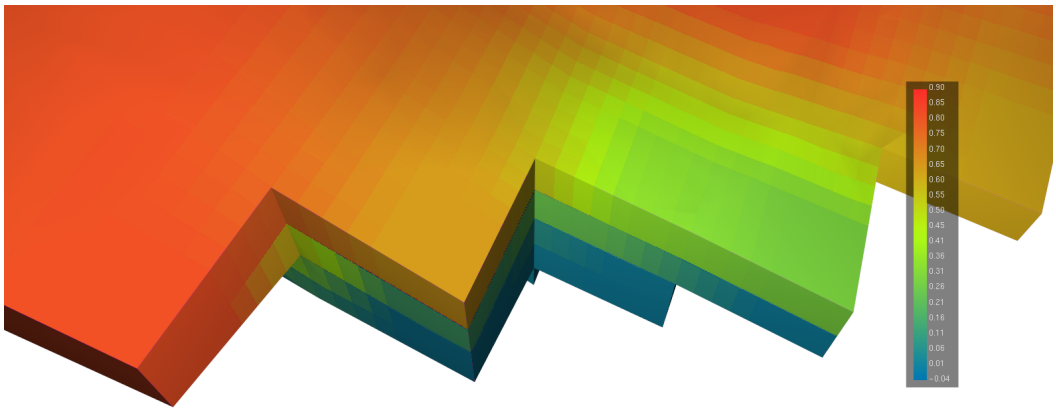


6.6(a): Propriedade não suavizada mapeada sobre a malha original

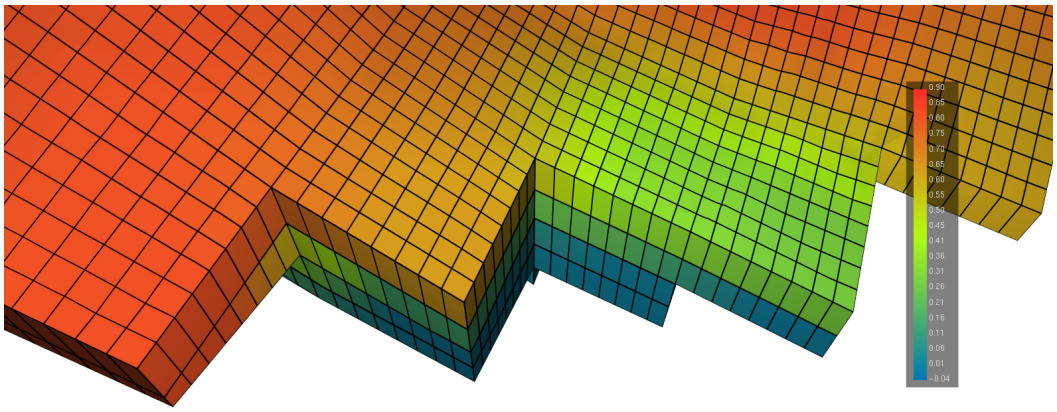


6.6(b): Malha simplificada utilizada nas figuras a seguir

Figura 6.6: Fotos de tela do mapeamento de textura de propriedade não suavizada em 3D.



6.7(a): Propriedade não suavizada mapeada sobre a malha simplificada



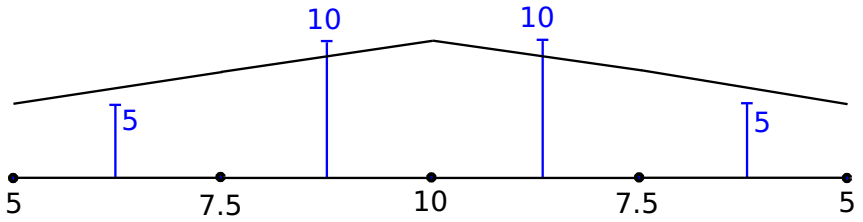
6.7(b): Propriedade não suavizada e grid original mapeados sobre a malha simplificada

Figura 6.7: Fotos de tela do mapeamento de textura de propriedade não suavizada em 3D.

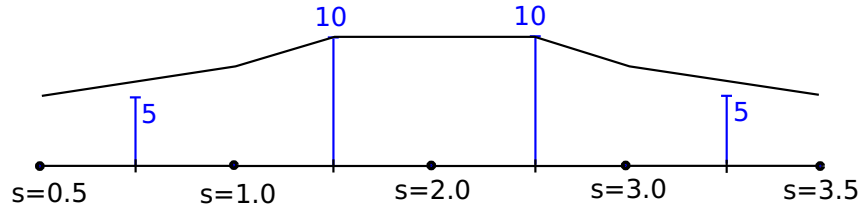
6.2.2
Mapeamento da Propriedade Suavizada

As propriedades do reservatório são dados associados às células da sua malha. É comum a visualização das propriedades suavizadas nos vértices dessa malha, não havendo um consenso sobre como calcular o valor do campo nos vértices da forma mais correta. O método usual de suavização define o valor da propriedade associada a um vértice como a média aritmética dos valores associados às células que possuem o dado vértice. O mapeamento da propriedade no interior das células é feito interpolando-se os valores das médias calculadas nos vértices. Dessa forma, assumindo que os dados originais estão definidos nos centros das células, este método não honra os valores das amostras nestas posições.

Analisemos em primeiro lugar um modelo 1D de reservatório, conforme ilustrado na Figura 6.8. O modelo da figura representa uma linha dividida em quatro segmentos de reta (equivalentes a células 1D), com um valor escalar associado a cada segmento. O método usual de suavização calcula o valor em cada vértice como sendo a média dos valores associados aos segmentos que possuem o dado vértice: os valores médios para os cinco vértices desse exemplo serão, da esquerda para a direita: 5, 7,5, 10, 7,5 e 5 (Figura 6.8(a)). A interpolação linear dos valores nos segmentos do meio resultará em uma variação linear entre os valores 10 e 7,5, possuindo o valor de 8,25 no centro destes segmentos, ao invés do valor definido originalmente para o segmento, que é igual a 10. O mesmo ocorre nos segmentos de borda. Isso mostra que o



6.8(a): Suavização usual via médias nos vértices



6.8(b): Suavização via mapeamento de textura

Figura 6.8: Análise 1D dos métodos de suavização do campo definido nos segmentos de reta.

uso de médias na suavização pode vir a esconder máximos ou mínimos locais ou globais dependendo do caso.

Nossa proposta de visualização de propriedade suavizada, neste caso, consiste em definir uma textura quatro *texels* com os valores escalares, 5, 10, 10 e 5, atribuindo a cada vértice uma coordenada de textura igual à média das coordenadas de textura nos centros dos segmentos que possuem o dado vértice. Utilizamos aqui a convenção em que o intervalo de coordenadas de textura é $[0, ni]$, com ni valendo 4 nesse exemplo. Dado o uso do filtro linear no mapeamento da textura e da interpolação das coordenadas de textura ao longo dos segmentos, obtemos do mapeamento os valores de propriedade corretos nos centros de todos os segmentos internos. Os valores obtidos nos vértices da malha também correspondem às médias dos valores associados aos segmentos que os contém. No entanto, assim como na suavização por médias, os valores associados aos segmentos de borda não serão honrados nos seus centros, como pode ser observado no exemplo da Figura 6.8(b).

Note que este método lida corretamente com as bordas do modelo (Figura 6.8(b)) e com falhas geológicas, que no caso 1D seriam dadas por descontinuidades na linha que representa o modelo, conforme ilustrado na Figura 6.9. O cálculo da coordenada de textura de vértices de borda e de vértices pertencentes a falhas considera apenas o segmento as que contém, atribuindo coordenadas no centro dos *texels* associados a esses segmentos. No caso de falhas, isso evita a consideração do *texel* que é vizinho na textura mas que se encontra do outro lado da falha. Dado o cálculo correto nos vértices e nos centros dos segmentos internos, assim como o tratamento adequado para bordas e falhas do modelo, consideramos o método aqui proposto como mais correto que o método usual de suavização por médias.

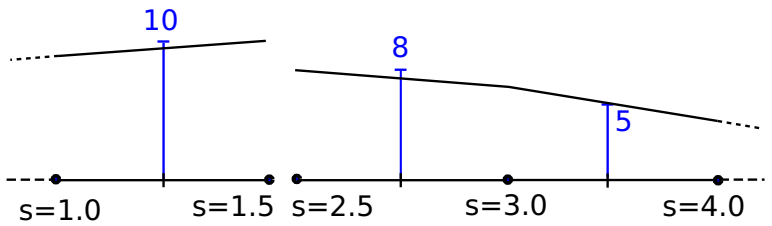


Figura 6.9: Suavização 1D via mapeamento de texturas no caso de falhas.

A extensão do problema para 2D trata de um modelo de reservatório descrito por uma malha de quadriláteros, sendo possível a presença de falhas. A extensão da nossa proposta para 2D define uma textura onde cada *texel* possui o valor da propriedade na célula correspondente e habilita o filtro bilinear do mapeamento da textura. Nós identificamos cinco tipos de casos para o cálculo

das coordenadas de textura atribuídas a um vértice nesse tipo de malha 2D, todos ilustrados na Figura 6.10.

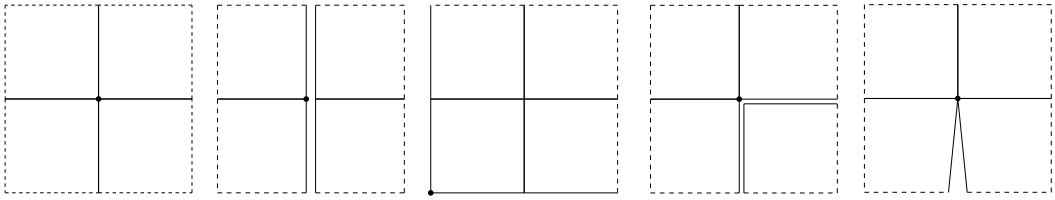


Figura 6.10: Casos para o cálculo de coordenadas das textura atribuídas a um vértice de uma malha 2D de reservatórios. Os vértices em questão se encontram em destaque, e as linhas tracejadas indicam a presença de outras células adjacentes.

A estratégia para o cálculo das coordenadas de textura para um vértice em 2D será uma extensão da utilizada em 1D, contendo uma análise do padrão de vizinhança de cada vértice. Esse cálculo é feito conforme o seguinte procedimento:

- obtenha o conjunto de células que contém o vértice;
- considerando apenas as células do conjunto, contabilize quantas células compartilham arestas com cada outra célula;
- obtenha as células com a maior contagem de arestas compartilhadas;
- a coordenada de textura atribuída ao vértice será dada pela média das coordenadas dos centros dos *texels* associados a essas células.

A Figura 6.11 apresenta a contagem de células adjacentes para os cinco casos 2D. Assim, no primeiro caso ilustrado na figura, a coordenada de textura do vértice será dada pela média das coordenadas dos centros dos quatro *texels* associados às células que possuem o vértice. Já no quarto caso, por exemplo, a coordenada será dada pelo centro do *texel* associado à célula à esquerda e acima.

As Figuras 6.12, 6.13, 6.14, 6.15 e 6.16 mostram cada caso 2D em detalhe. Elas acompanham uma malha de exemplo e uma ilustração da

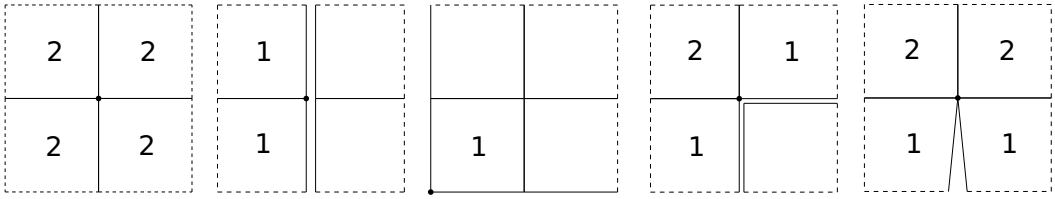


Figura 6.11: Contagens do número de células adjacentes que possuem os vértices em questão para os cinco casos 2D.

coordenada escolhida para o vértice no espaço paramétrico da textura. Elas também mostram exemplos da renderização da propriedade não suavizada e da mesma propriedade suavizada pela forma usual, utilizando médias, e pelo nosso método, utilizando mapeamento de textura.

O primeiro caso 2D (Figura 6.12) ilustra o cálculo de coordenadas de textura para vértices internos da malha. O algoritmo proposto garante, assim como no caso 1D, que a média feita pelo filtro bilinear fará a suavização de forma a honrar o valor das amostras no centro de todas as células *internas*.

O segundo caso (Figura 6.13) é de um vértice que se encontra no meio de uma falha geológica ou na fronteira externa do modelo, pertencendo a apenas duas células. A interpolação bilinear fará corretamente a interpolação entre os valores associados às duas células que contém o vértice na direção vertical. No caso de falha geológica, as células do outro lado da falha não serão consideradas, conforme o esperado. Assim como na suavização usual, a interpolação na direção perpendicular da falha não honrará os valores nos centros das células de borda.

No terceiro caso o vértice faz parte apenas de uma célula (Figura 6.14). A coordenada de textura atribuída a ele será o centro do *texel* associado à única célula que o contém, forçando que a suavização nas duas direções seja similar à suavização feita nos segmentos de borda em 1D, que foi ilustrada na Figura 6.8.

A Figura 6.15 ilustra o quarto caso, no qual o vértice se encontra em uma quina. A suavização usual considera as três células que possuem o vértice. Já o algoritmo proposto evita a consideração da célula à direita e abaixo na figura, dada a escolha de uma coordenada de textura no centro do *texel* da célula à esquerda e acima. A diferença no resultado visual pode ser observada nas Figuras 6.15(d) e 6.15(e).

O quinto e último caso 2D está ilustrado na Figura 6.16, onde o vértice se encontra no início de uma falha geológica e pertence a um conjunto de quatro células. Nossa proposta posiciona a coordenada de textura entre os centros dos *texels* associados às duas células de cima na figura.

A extensão desse método para 3D, que é onde reside o nosso problema, é feita utilizando a mesma estratégia da extensão de 1D para 2D, substituindo dessa vez a consideração de arestas pela consideração de faces. O algoritmo, que pode necessitar de mais de uma iteração para a decisão da coordenada de textura final, é composto pelos seguintes passos:

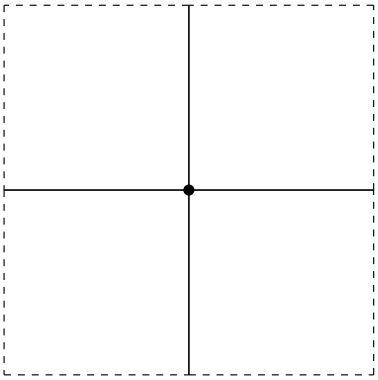
– obtenha o conjunto de células que contém o vértice;

– repita continuamente os seguintes passos:

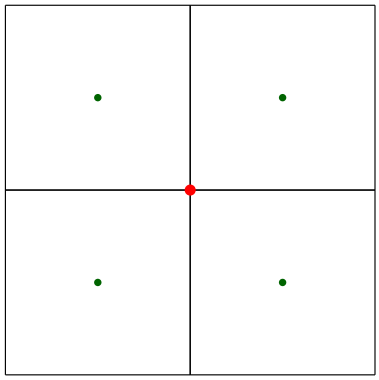
  - considerando apenas as células no conjunto, contabilize quantas células compartilham faces com cada outra célula;
  - obtenha as células com a maior contagem de faces compartilhadas e exclua as demais células do conjunto;
  - caso o conjunto não tenha mudado desde o passo anterior, pare.

- a coordenada de textura final será a média das coordenadas dos centros dos *voxels* associados às células no conjunto final.

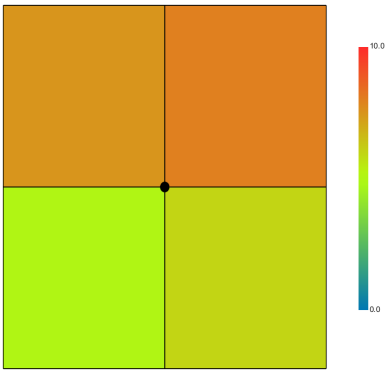
A Figura 6.17 ilustra os resultados visuais obtidos no mapeamento da textura de propriedade 3D: a Figura 6.17(a) mostra a malha original do reservatório com o desenho de uma propriedade utilizando o método proposto, a Figura 6.17(b) mostra uma malha simplificada e a Figura 6.17(c) mostra a propriedade mapeada sobre a malha simplificada. Como interpolamos coordenadas de textura e não valores de propriedades ao longo das primitivas maiores, o resultado visual obtido é praticamente idêntico ao obtido com o modelo original.



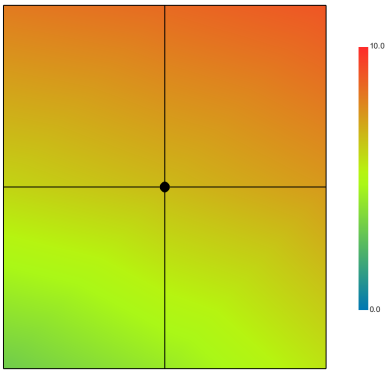
6.12(a): Malha e vértice em questão



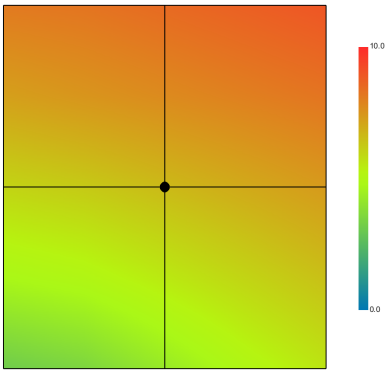
6.12(b): Coordenada calculada no espaço paramétrico da textura



6.12(c): Dados originais por célula

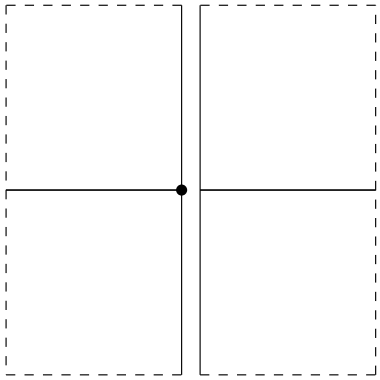


6.12(d): Suavização usual

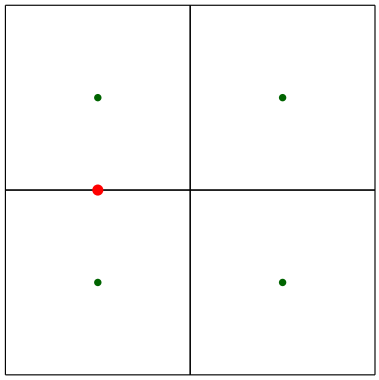


6.12(e): Suavização proposta

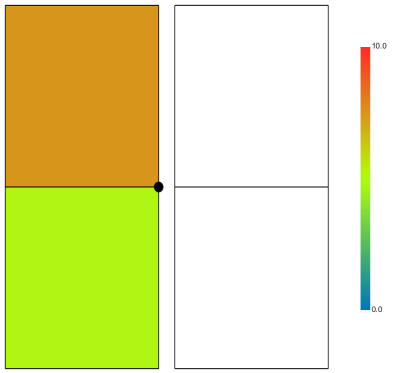
Figura 6.12: Ilustração do primeiro caso 2D de suavização via textura de propriedade. (a) Malha de exemplo, com o vértice em questão em destaque (b) Ilustração dos *texels* da textura em verde e da coordenada de textura escolhida em vermelho, todos no espaço paramétrico da textura. (c) Dados originais definidos nas células (d) Resultado visual obtido utilizando a suavização usual via médias (e) Resultado visual obtido utilizando a suavização utilizando mapeamento de textura.



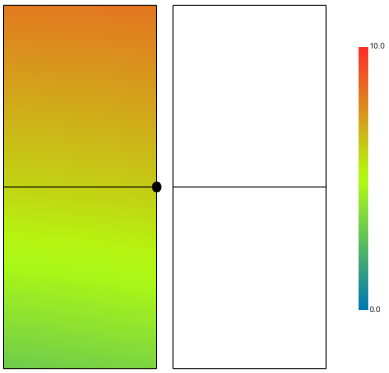
6.13(a): Malha e vértice em questão



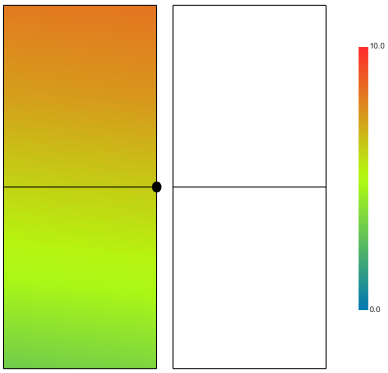
6.13(b): Coordenada calculada no espaço paramétrico da textura



6.13(c): Dados originais por célula

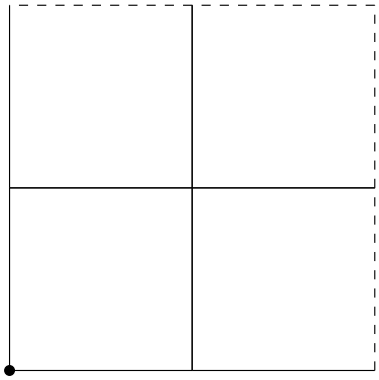


6.13(d): Suavização usual

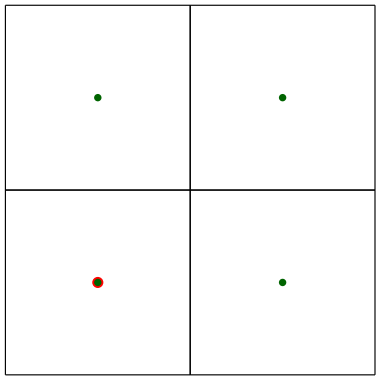


6.13(e): Suavização proposta

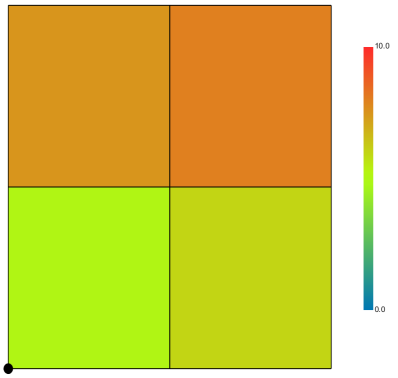
Figura 6.13: Ilustração do segundo caso 2D de suavização via textura de propriedade. (a) Malha de exemplo, com o vértice em questão em destaque (b) Ilustração dos *texels* da textura em verde e da coordenada de textura escolhida em vermelho, todos no espaço paramétrico da textura. (c) Dados originais definidos nas células (d) Resultado visual obtido utilizando a suavização usual via médias (e) Resultado visual obtido utilizando a suavização utilizando mapeamento de textura.



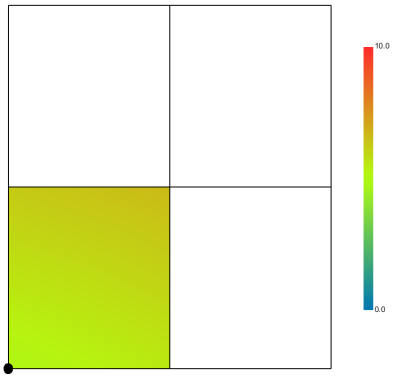
6.14(a): Malha e vértice em questão



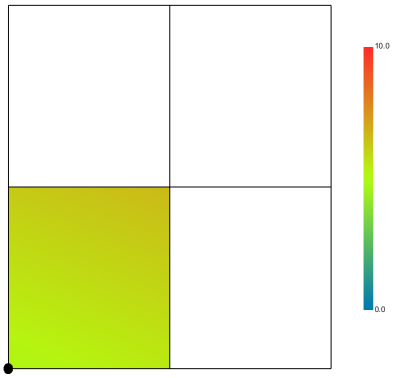
6.14(b): Coordenada calculada no espaço paramétrico da textura



6.14(c): Dados originais por célula

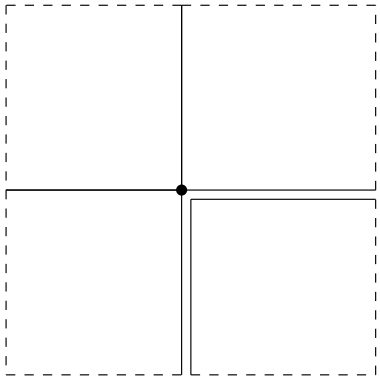


6.14(d): Suavização usual

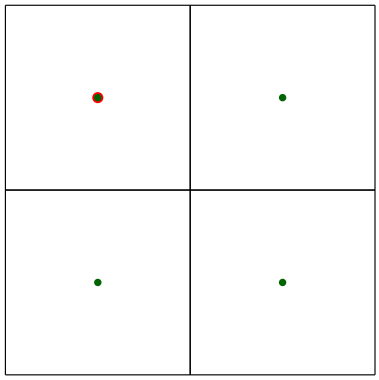


6.14(e): Suavização proposta

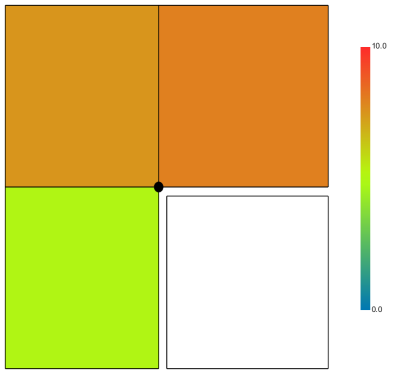
Figura 6.14: Ilustração do terceiro caso 2D de suavização via textura de propriedade. (a) Malha de exemplo, com o vértice em questão em destaque (b) Ilustração dos *texels* da textura em verde e da coordenada de textura escolhida em vermelho, todos no espaço paramétrico da textura. (c) Dados originais definidos nas células (d) Resultado visual obtido utilizando a suavização usual via médias (e) Resultado visual obtido utilizando a suavização utilizando mapeamento de textura.



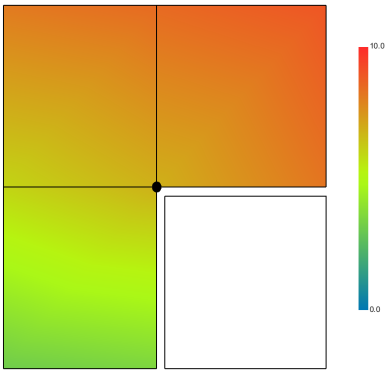
6.15(a): Malha e vértice em questão



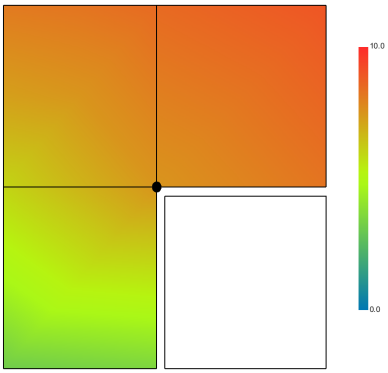
6.15(b): Coordenada calculada no espaço paramétrico da textura



6.15(c): Dados originais por célula

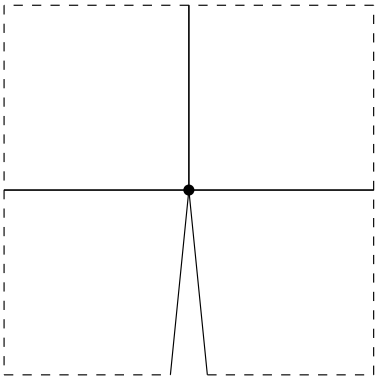


6.15(d): Suavização usual

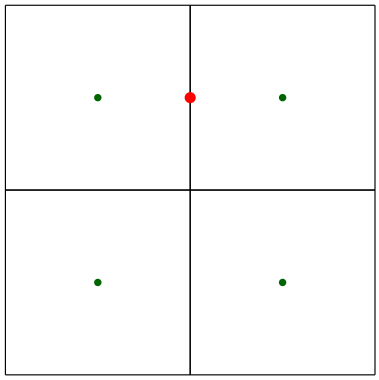


6.15(e): Suavização proposta

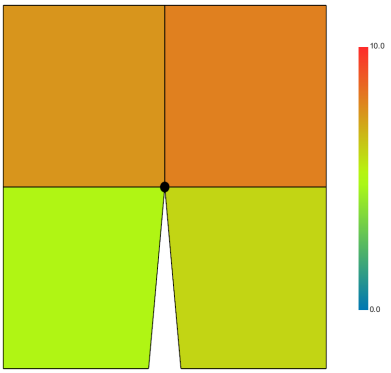
Figura 6.15: Ilustração do quarto caso 2D de suavização via textura de propriedade. (a) Malha de exemplo, com o vértice em questão em destaque (b) Ilustração dos *texels* da textura em verde e da coordenada de textura escolhida em vermelho, todos no espaço paramétrico da textura. (c) Dados originais definidos nas células (d) Resultado visual obtido utilizando a suavização usual via médias (e) Resultado visual obtido utilizando a suavização utilizando mapeamento de textura.



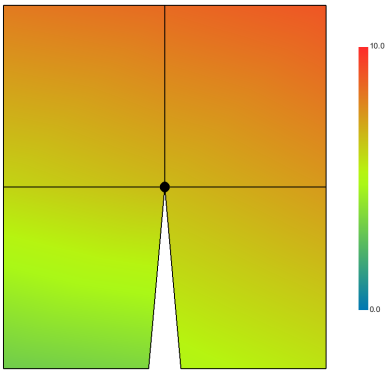
6.16(a): Malha e vértice em questão



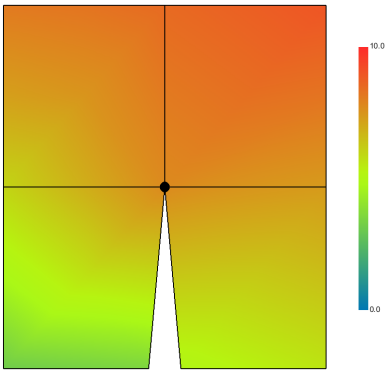
6.16(b): Coordenada calculada no espaço paramétrico da textura



6.16(c): Dados originais por célula

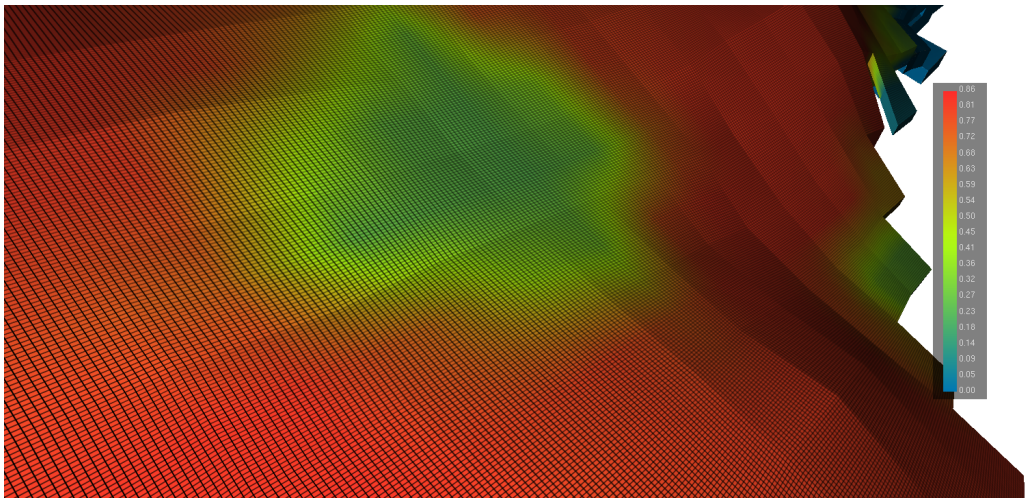


6.16(d): Suavização usual

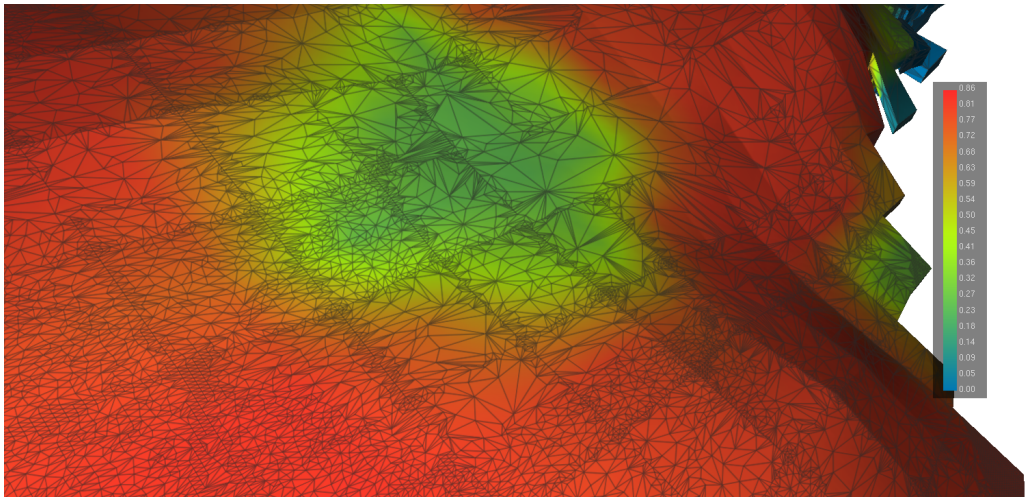


6.16(e): Suavização proposta

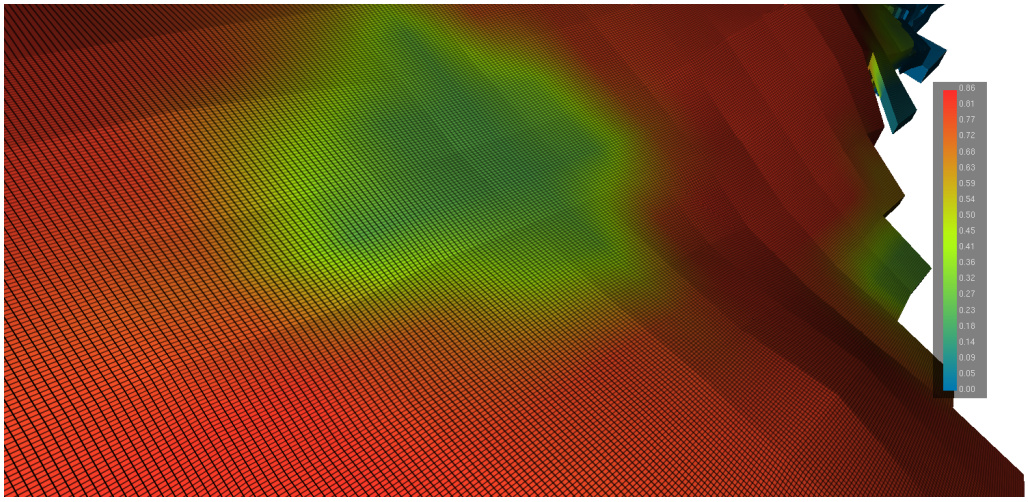
Figura 6.16: Ilustração do quinto caso 2D de suavização via textura de propriedade. (a) Malha de exemplo, com o vértice em questão em destaque (b) Ilustração dos *texels* da textura em verde e da coordenada de textura escolhida em vermelho, todos no espaço paramétrico da textura. (c) Dados originais definidos nas células (d) Resultado visual obtido utilizando a suavização usual via médias (e) Resultado visual obtido utilizando a suavização utilizando mapeamento de textura.



6.17(a): Propriedade mapeada sobre o modelo original



6.17(b): Malha do modelo simplificado



6.17(c): Propriedade mapeada sobre o modelo simplificado

Figura 6.17: Fotos de tela do mapeamento da propriedade suavizada em 3D.

6.3
Resultado Experimental

Nós avaliamos o impacto da inclusão do mapeamento do *grid* e da propriedade no desempenho do nosso visualizador com multi-resolução. O desempenho do nosso sistema foi medido no mesmo equipamento em que foram feitos os experimentos do capítulo anterior.

A Figura 6.18 mostra o desempenho em termos de taxas de renderização utilizando diferentes configurações de mapeamento, que foram medidas ao longo do caminho de câmera descrito no capítulo anterior e com um limite de erro fixo para o modelo B100. Foram medidos os desempenhos dos seguintes modos de visualização: sem mapeamentos; mapeando somente o *grid* original sobre a malha simplificada; mapeando o *grid* e uma propriedade não suavizada; e mapeando o *grid* e uma propriedade suavizada. Os resultados mostram que há um impacto na taxa de renderização quando incluimos os diferentes mapeamentos, especialmente quando é incluído o mapeamento de propriedades, que faz uso de uma textura 3D. Apesar disto, consideramos os resultados satisfatórios em termos de desempenho, dada a qualidade das imagens mostradas anteriormente.

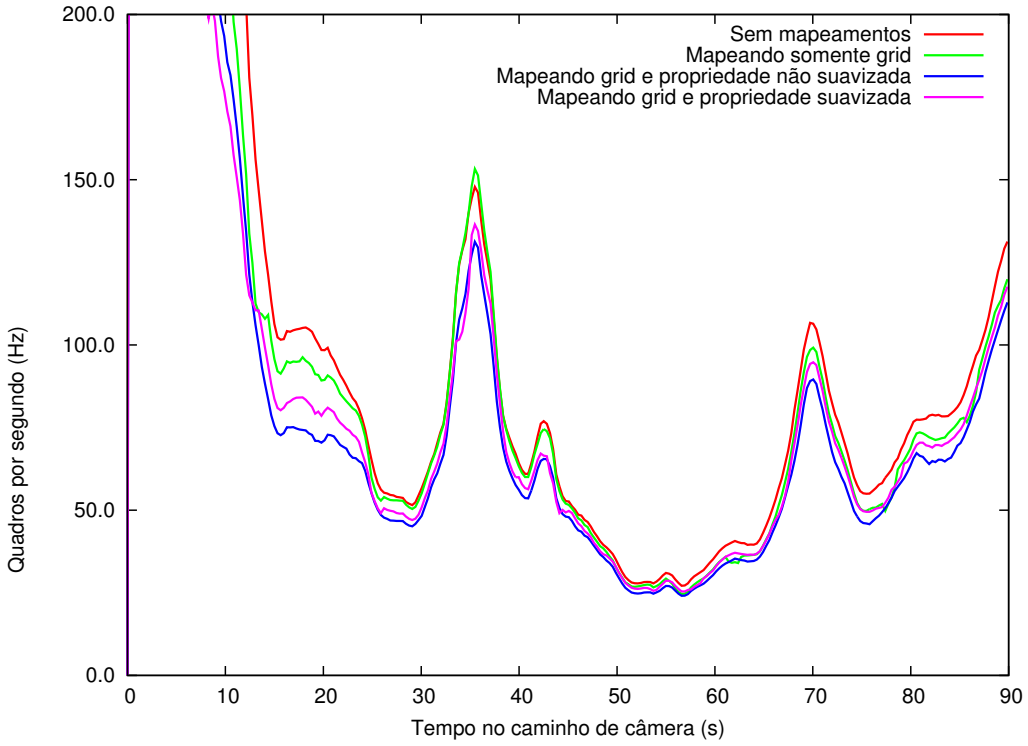


Figura 6.18: Taxas de renderização obtidas com diferentes configurações de mapeamento de dados do modelo original sobre o modelo de multi-resolução, modelo B100.

6.4 Conclusão

O trabalho apresentado neste capítulo apresentou propostas para o mapeamento dos dados associados à malha original do reservatório, como o *grid* e as propriedades associadas às células sobre o suporte geométrico do reservatório, o que foi feito pelo uso de mapeamento de texturas.

Conforme foi mostrado ao longo do capítulo, a qualidade das imagens obtidas com o mapeamento desses dados sobre as malhas simplificadas foi bastante satisfatória, sendo praticamente iguais às imagens geradas com o modelo original. Note que as técnicas propostas utilizam no total apenas dois conjuntos de coordenadas de textura por vértice: as coordenadas topológicas i, j e k são utilizadas no mapeamento do *grid* original e da propriedade não suavizada e as coordenadas de propriedade, que são de certa forma deslocamentos das coordenadas topológicas i, j e k, são utilizadas no mapeamento da propriedade suavizada. Os métodos propostos possuem fácil integração com esquemas de multi-resolução, pois apenas requerem a inclusão dessas coordenadas de textura como atributos de vértice. Elas serviram para tornar a estrutura de multi-resolução independente dos valores de propriedade, garantindo o reuso dessa estrutura em múltiplas simulações do mesmo modelo, o que é muito importante dados os longos tempos de pré-processamento no uso de técnicas de multi-resolução para modelos de tamanha magnitude.

O método aqui proposto possui no entanto uma forte limitação causada pelo uso de uma textura 3D com $ni \times nj \times nk$ *voxels*: o seu grande uso de memória de vídeo. Esse requisito pode até impossibilitar que alguns modelos sejam visualizáveis. Um possível trabalho futuro seria a investigação de técnicas de partição da textura em blocos e de gerência da memória de vídeo ocupada por esses blocos. Isso faria com que cada região triangular da hierarquia só acessasse uma pequena porção do conjunto com todos os blocos de textura. A investigação de esquemas de multi-resolução no espaço de textura é vista como bem complicada, dado que é necessário respeitar falhas geológicas ao calcular médias de valores de propriedade.

As propostas foram projetadas com o objetivo principal de dar apoio à estrutura de multi-resolução, porém acreditamos que ela também possa ser aplicada a modelos originais de médio porte. Sua grande vantagem é o uso de um conjunto pequeno de dados adicionais ao suporte geométrico, sem a necessidade de replicação de vértices, o que atualmente não é possível por exemplo na visualização do reservatório original com a propriedade não suavizada (por célula). Essa redução no uso de memória pelo armazenamento dos vértices deve ser comparada com o uso extra de memória de vídeo por

conta da textura 3D. A comparação experimental dos métodos usuais para a visualização de modelos de médio porte e dos métodos aqui propostos, tanto em termos de uso de memória quanto em termos de desempenho, permanece como trabalho futuro.

Outro trabalho futuro seria investigar uma forma melhor de lidar com a suavização em células de borda, talvez fazendo o uso de *shaders* para a correção da interpolação. A técnica apresentada tem a vantagem de requisitar apenas uma textura 3D com os dados de propriedade e coordenadas de textura por vértice. O uso de *shaders* tem que levar em consideração que o armazenamento, por exemplo, de informações de falhas, na memória da GPU forçaria ainda mais a limitação que nossa técnica apresenta no uso desse escasso recurso computacional.