5
Visualização com Multi-resolução

Nesse capítulo é feita uma proposta para a visualização de malhas de reservatórios naturais de petróleo com multi-resolução. Em primeiro lugar será apresentada nossa proposta de algoritmo para a simplificação de malhas de reservatório. A estrutura hierárquica de multi-resolução utilizada será apresentada na seção seguinte, em conjunto com a organização dos dados em memória secundária. Por último, será descrita a visualização com multi-resolução dependente da posição do observador e serão apresentados os resultados experimentais obtidos.

5.1
Simplificação de Malhas de Reservatório

Esta seção apresenta nossa proposta de algoritmo para a simplificação de malhas de reservatório. O objetivo é gerar malhas com um número cada vez menor de elementos hexaédricos a partir da malha original do reservatório. A malha resultante deve manter as características da malha original na medida do possível:

- deve ser composta apenas por elementos hexaédricos, permitindo o uso da maioria dos algoritmos de visualização conhecidos sem a necessidade de adaptação;
- deve manter uma boa aproximação da geometria da malha original, especialmente próximo às fronteiras do modelo e a falhas geológicas;
- deve manter uma boa aproximação dos atributos associados à malha original;
- deve permitir a visualização das camadas em separado.

Um objetivo secundário porém importante é obter elementos hexaédricos com uma boa forma. Um hexaedro possui boa forma se os ângulos internos entre suas arestas forem todos próximos de 90 graus.

Blacker e Stephenson (8) definiram o operador de fechamento de elemento (*element close*) para a simplificação de malhas de quadriláteros. Esse operador une os dois vértices opostos de um quadrilátero, reduzindo o número de quadriláteros da malha em um, conforme ilustrado na Figura 5.1. Staten et al. (52) consideraram a extensão desse operador para malhas de hexaedros, que remove *colunas* de elementos, definidas como uma sequência de elementos hexaédricos adjacentes pelas suas faces de topo e de base, conforme ilustrado na Figura 5.2.

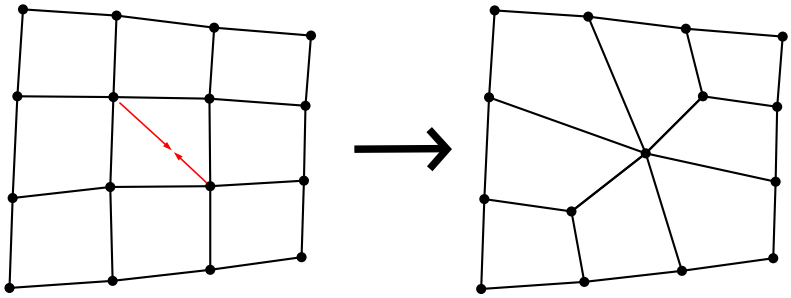


Figura 5.1: Operador de fechamento de elemento para malhas de quadriláteros.

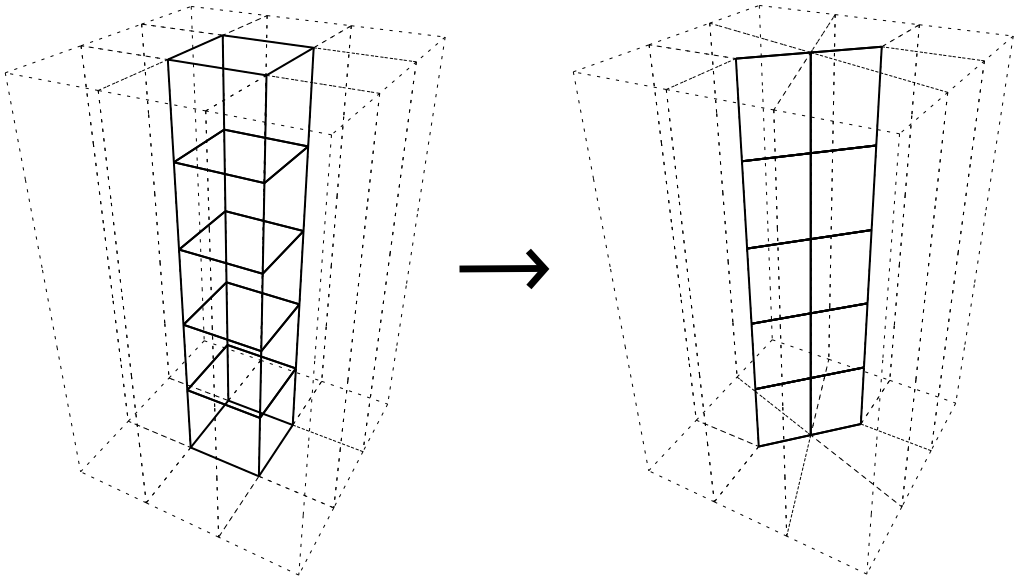


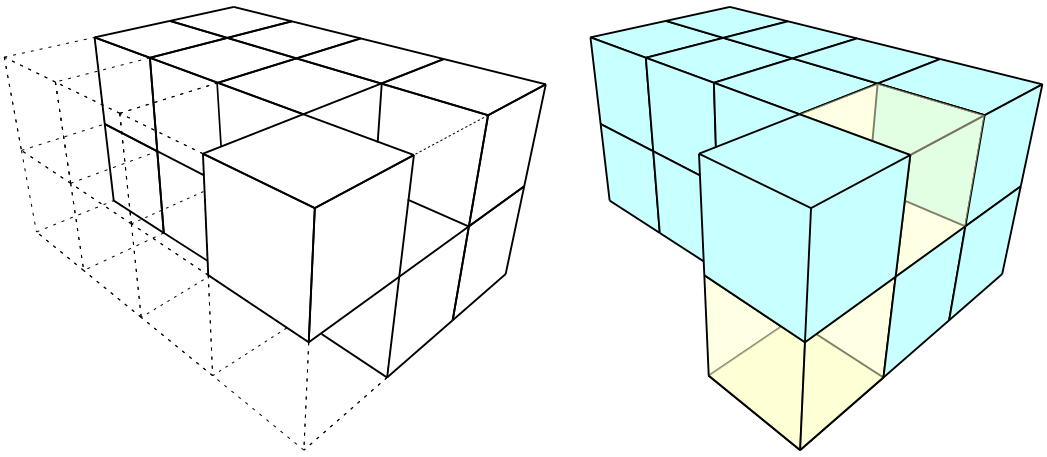
Figura 5.2: Operador de colapso de coluna para malhas de hexaedros.

5.1.1 Malhas Utilizadas na Simplificação

O operador de colapso de coluna se encaixa bem nos requisitos topológicos da simplificação de malhas de reservatório acima enumerados. As colunas nesse tipo de malha são definidas como o conjunto de células que possuem as mesmas coordenadas topológicas i e j no modelo original. Esse operador também permite a visualização de camadas em separado, que permite um entendimento melhor do fluxo dos fluidos no interior do reservatório ao longo da simulação.

Algumas colunas do modelo de reservatório são incompletas devido à presença de células inativas, como representado na Figura 5.3(a). Nosso algoritmo propõe inserir células inativas no modelo a ser simplificado para que todas as suas colunas sejam completas: cada coluna que possui ao menos uma célula ativa é tornada completa pela inserção das células inativas com o mesmo i e j no modelo original, conforme ilustrado na Figura 5.3(b). Esse

procedimento permite que os colapsos sejam efetuados em um modelo onde todas as colunas possuem nk células, simplificando as decisões do algoritmo.



5.3(a): Modelo original e células inativas

5.3(b): Colunas do modelo completadas

Figura 5.3: Ilustração das colunas do modelo de reservatório sendo completadas com células inativas. É mantida uma marcação de quais células da malha são ativas (células em azul na figura à direita).

5.1.2 Operadores de Simplificação

Nosso algoritmo de simplificação se baseia em um conjunto de quatro operadores de colapso de coluna, todos ilustrados com uma vista de cima nas Figuras 5.4, 5.5, 5.6 e 5.7. Relembramos que as colunas são compostas por todas as células com a mesma coordenada topológica i e j, as quais são adjacentes umas às outras pelas suas faces de topo e de base.

O primeiro operador de colapso opera colunas *internas* da malha, onde os quatro vértices da face de topo e da face de base são internos, não se encontrando na borda lateral do modelo. Esse operador, introduzido por Staten et al. (52), une vértices opostos das faces de topo e de base de cada célula (Figura 5.2). Ao efetuar um colapso, faces laterais adjacentes são unidas em apenas uma. O colapso encontra-se ilustrado com uma vista de cima na Figura 5.4. Nele, os vértices opostos v_1 e v_2 são unidos para formar um novo vértice $\bar{v}$. Duas direções de colapso são possíveis para cada coluna interna. Nosso algoritmo propõe selecionar uma dessas direções com base no conceito de *valência lateral* do vértice, que é o número de faces laterais que possuem o dado vértice, equivalente ao número de vizinhos do vértice nas ilustrações aqui apresentadas. A escolha da direção utilizada é feita de acordo com o seguinte critério: identifica-se o vértice com a menor valência lateral dentre os

quatro vértices da face de topo da coluna. Elege-se a direção que possui esse vértice dentre os vértices colapsados v_1 e v_2. Caso haja mais de um vértice com a menor valência em direções diferentes, o critério passa a ser a menor valência para o novo vértice $\bar{v}$: se $val(v)$ é a valência lateral do vértice v, então $val(\bar{v}) = val(v_1)+val(v_2)-2$. Caso haja um novo empate, a direção é escolhida aleatoriamente. A estratégia proposta ajuda a manter valências laterais baixas, tendendo a formar malhas com boa qualidade, além de acelerar o processo de simplificação, visto que todos os cálculos associados a uma das direções, como avaliações de erro, validade, etc são evitados. As valências laterais dos vértices v_{r1} e v_{r2}, isto é, vértices das faces não colapsados (Figura 5.4), são reduzidas em um após o colapso, fato que será considerado adiante.

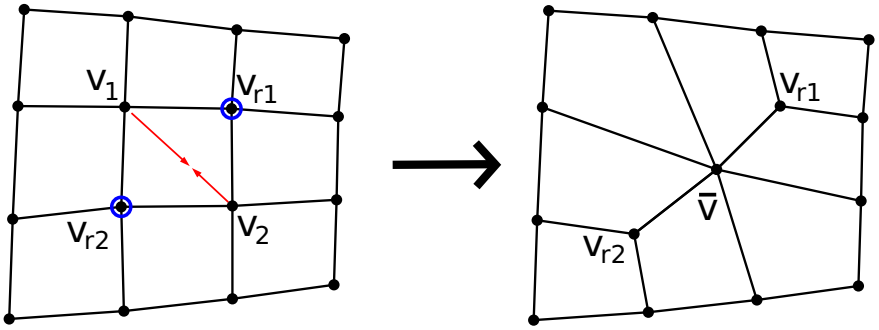


Figura 5.4: Colapso de colunas internas (visão 2D de cima). Os vértices que permanecem na malha estão ressaltados em azul.

O segundo operador de colapso é uma extensão do primeiro operador para colunas onde um ou dois vértices da face de topo e da face de base encontram-se na borda lateral do modelo (Figura 5.5). Os vértices v_1 e v_2 são unidos para formar o vértice $\bar{v}$, que é posicionado na borda. Assim como no primeiro tipo de colapso, há duas direções possíveis. A mesma regra de seleção de direção descrita acima é utilizada, e as valências laterais dos vértices v_{r1} e v_{r2} ilustrados na figura também são reduzidas em um após o colapso.

O terceiro operador de colapso é nossa proposta para operar colunas com duas faces laterais de borda consecutivas, as quais possuem três vértices na borda lateral do modelo em cada face de topo e de base (Figura 5.6). Esse tipo de configuração surge, por exemplo, após o segundo tipo de colapso: vide a célula à esquerda e abaixo no resultado do colapso ilustrado na Figura 5.5. O operador une os três vértices na borda formando o vértice $\bar{v}$. O vértice v_r ilustrado na figura tem sua valência lateral reduzida em um após o colapso.

Introduzimos ainda um quarto operador de colapso (Figura 5.7), que opera em colunas com uma face lateral interna e três faces laterais na borda, possuindo todos os vértices na borda lateral do modelo. Esse tipo de

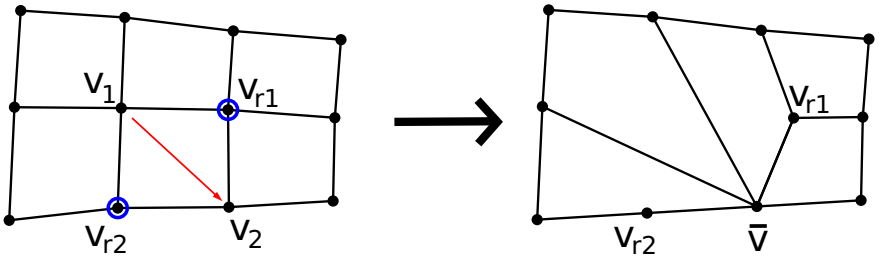


Figura 5.5: Colapso de colunas com um ou dois vértices na borda (visão 2D de cima). Os vértices que permanecem na malha estão ressaltados em azul.

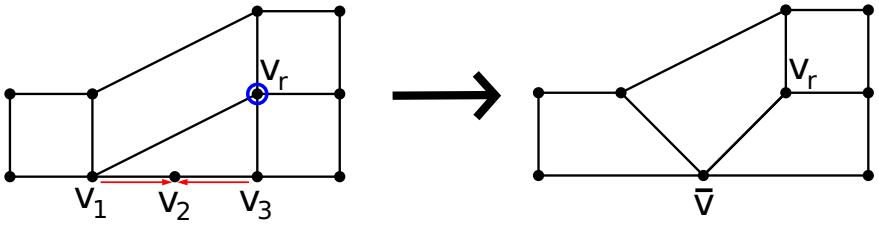
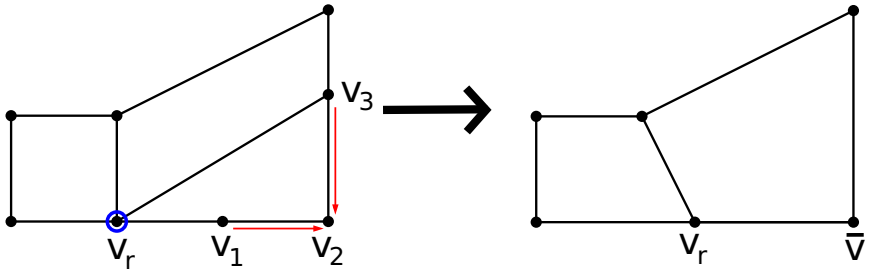


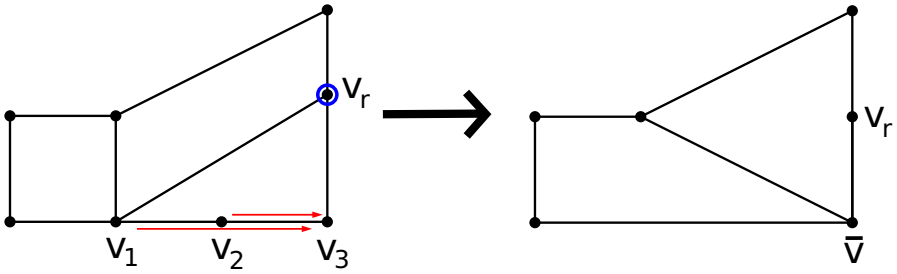
Figura 5.6: Proposta de colapso de colunas com três vértices na borda (visão 2D de cima). O vértice que permanece na malha está ressaltado em azul.

configuração surge após o segundo e o terceiro tipo de colapso. Dada a face lateral interna, duas opções de colapso são possíveis ao se selecionar um dos vértices para permanecer na malha (v_r nas ilustrações das duas opções) e unindo os outros três vértices (v_1, v_2 e v_3). A posição para o novo vértice $\bar{v}$ deve ser escolhida entre uma das posições de v_1, v_2 e v_3. A seleção de qual será v_r e qual a posição de $\bar{v}$ dentre as posições de v_1, v_2 e v_3 deve ser feita com base num critério de erro geométrico de aproximação, que será descrito na próxima seção.

Os vértices da coluna colapsada que permanecem na malha (marcados em azul nas ilustrações acima) têm sua valência lateral reduzida em um após cada colapso. Se a valência lateral de um vértice interno se torna dois, ele passa a fazer parte de um *doublet*, que é uma configuração conhecida na literatura onde dois elementos hexaédricos compartilham duas faces laterais consecutivas, conforme ilustrado com uma visão de cima na Figura 5.8 à esquerda. Ao analisarmos em 2D, o vértice v_d possui dois ângulos internos que somam 360 graus. Esta configuração pode conter uma ou até duas células com ângulos internos muito grandes, portanto com má forma. Assim, como em trabalhos anteriores (21, 54), optamos por eliminar essa configuração assim que detectada, o que é feito pela remoção de qualquer uma das colunas que formam o *doublet*, por exemplo, a que contém v_d e v_r na ilustração. Os vértices v_d são removidos e os novos vértices $\bar{v}$ substituirão os vértices v_r. Esta operação também pode gerar novos doublets, afinal os vértices que eram adjacentes a



5.7(a): Opção 1



5.7(b): Opção 2

Figura 5.7: Proposta de colapso de colunas com quatro vértices na borda (visão 2D de cima). À esquerda, a primeira opção de colapso, à direita, a segunda opção. O vértice que permanece na malha está ressaltado em azul.

v_d também têm sua valência reduzida em um. Portanto, a detecção e remoção de *doublets* é repetida até que não haja *doublets* a ser removidos. O mesmo tratamento foi experimentado em vértices de borda com valência lateral 2, porém o ganho em qualidade de malha não compensou o aumento no erro de simplificação.

5.1.3 Algoritmo de Simplificação

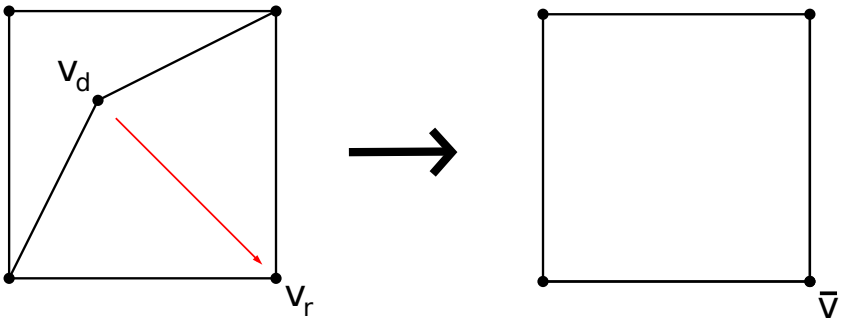


Figura 5.8: Remoção de *doublets* (visão 2D de cima).

Medição do Erro de Simplificação

O colapso de uma coluna introduz um erro geométrico, que deve ser mensurado para que seja possível gerar boas aproximações da malha original. A métrica de erro quádrico apresentada por Garland e Heckbert (24) foi adaptada para a manipulação de colunas de células hexaédricas. Calcula-se matrizes quádricas associadas a cada vértice da malha original, de acordo com os requisitos de simplificação enumerados anteriormente.

Dado o objetivo de se poder separar as camadas do reservatório, considera-se todas as faces de topo e de base de células ativas no cálculo de matrizes quádricas (faces de tipo 1 na Figura 5.9). O plano associado a cada uma dessas faces é acumulado com peso 1 nas quádricas de cada um dos seus vértices[1]. Já as faces laterais contribuem com as quádricas de seus vértices se forem *externas* e pertencerem a uma célula ativa. Existem três tipos de faces laterais externas:

- face compartilhada entre uma célula ativa e uma célula inativa (faces de tipo 2 na Figura 5.9);
- face pertencente a uma falha geológica, não sendo compartilhada entre duas células adjacentes na grade topológica (tipo 3);
- face pertencente à fronteira externa do modelo, estando nos limites da grade topológica, onde não existem algumas células vizinhas (tipo 4).

Os planos associados às faces laterais são acumulados nas quádricas de cada um dos seus vértices com um peso grande. Isso guia nosso processo de simplificação

[1]É calculado em cada vértice da face o plano cuja normal é dada pelo produto vetorial entre os vetores das arestas ligadas ao vértice e que contém o vértice em questão.

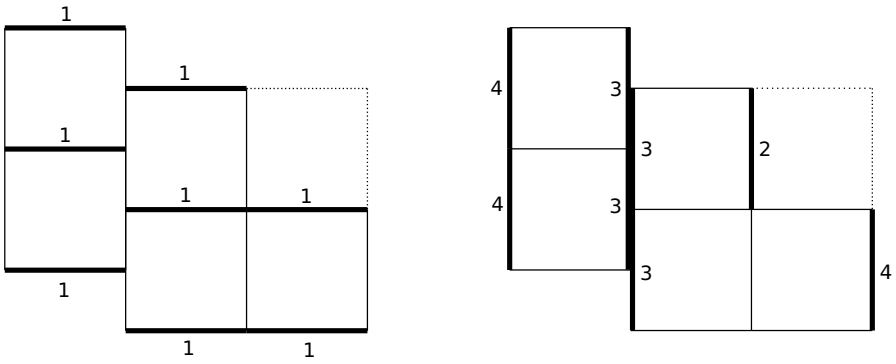


Figura 5.9: Visão lateral de um reservatório mostrando os tipos de faces consideradas no cálculo de matrizes quádricas. A célula mais à direita e mais ao topo da malha é inativa. Tipos de faces consideradas: 1) faces de topo e de base; 2) faces laterais compartilhadas entre células ativas e inativas; 3) faces laterais em falhas geológicas; 4) faces laterais na fronteira externa do modelo.

para respeitar as fronteiras das células ativas e as falhas geológicas do modelo o máximo possível.

O colapso de colunas une um conjunto de vértices ao longo das diversas camadas do modelo. A matriz quádrica associada a um novo vértice é calculada como a soma das matrizes quádricas de cada vértice substituído, assim como no trabalho original de Garland e Heckbert (24). O custo associado a cada colapso de coluna é então calculado como o maior erro cometido ao se unir ou mover os vértices de cada uma das células de uma coluna. O erro cometido em cada vértice é calculado como no trabalho original: $v^T Q v$, onde v é a posição do vértice e Q é a sua matriz quádrica associada.

Suavização

Diversos trabalhos na área de geração de malhas ressaltam a importância de se suavizar malhas, visando a obtenção de elementos com uma boa forma (51, 21, 48, 54). No nosso caso, é ideal gerar hexaedros com ângulos mais próximos de ângulos retos quanto for possível. Idealmente, a malha inteira deveria ser suavizada após efetuarmos cada colapso, entretanto isso seria extremamente custoso no nosso caso, onde cada coluna possui nk células.

Nossa proposta introduz formas locais de suavização a cada passo do algoritmo. Diferentemente de trabalhos anteriores, nosso algoritmo não decide as posições dos novos vértices se baseando meramente em minimizações de erro a partir das matrizes quádricas. Ao invés disso, a cada colapso de coluna inclui-se os novos vértices e também os vértices que permanecem na malha em um processo local de suavização nas direções x e y. Após determinadas essas posições, é feito um processo de minimização de erro apenas na direção z.

O processo de suavização tem como base a suavização Laplaciana (23) dos vértices envolvidos em uma operação de colapso. Inicialmente é feito o cálculo da posição inicial dos novos vértices de um colapso como a média das posições dos vértices sendo unidos. Os atributos de vértice também são inicializados da mesma forma. Em seguida, os novos vértices e os vértices remanescentes de cada face de topo e de base da coluna em questão são incluídos no processo de suavização se respeitarem a seguinte restrição: não podem pertencer a nenhuma face lateral externa. Essa restrição foi incluída para que seja respeitada a borda do modelo.

A suavização é feita em um processo iterativo, onde os N vértices participantes são visitados em sequência, repetidamente:

– calcule a nova posição e os novos atributos para o vértice i como média dos valores de todos os vértices adjacentes lateralmente;

– caso nenhum vértice tenha tido sua posição ou seus atributos alterados após N iterações, pare;
– caso contrário, siga para o próximo vértice: vértice $i + 1$ caso $i < N$, vértice 1 caso contrário.

No final do processo de suavização teremos cada vértice posicionado na média dos seus vizinhos. As ilustrações do primeiro, segundo e terceiro operador de colapso feitas anteriormente (Figuras 5.4, 5.5 e 5.6) mostram o resultado desse processo de suavização nas posições dos novos vértices e vértices marcados em azul.

O processo de cálculo das posições dos vértices é finalizado pela minimização do erro cometido na direção z. Assim como feito no trabalho original por Garland e Heckbert (24), a posição ótima para um vértice é calculada a partir da derivada parcial do erro quádrico nos eixos. A expressão do erro quádrico é dada por $\Delta(v = \{x, y, z\}) = v^T Q v = q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 + 2q_{23}yz + 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44}$, onde q_{ij} é o termo da matriz quádrica Q na linha i e coluna j. Como apenas minimizamos o erro na direção z, obtemos a derivada parcial do erro apenas nessa direção: $\delta\Delta/\delta z = 2q_{13}x + 2q_{23}y + 2q_{33}z + 2q_{34}$. Dado que as coordenadas x e y estão fixas, a coordenada z de erro mínimo na posição x e y é dada pelo ponto onde a derivada parcial é zero: $z = \frac{-q_{13}x\ -\ q23y\ -\ q_{34}}{q_{33}}$, com q_{33} diferente de zero.

Após calculadas as posições de cada vértice no topo e na base de cada célula, é detectado se as células adjacentes tiveram as suas faces invertidas, o que é conhecido como um *fold-over*. Um candidato a colapso é invalidado caso seja detectada a inversão de alguma face.

Nossa proposta opta por suavizar ao invés de escolher posições para os novos vértices com base apenas em uma minimização do erro quádrico cometido. Nossos resultados experimentais demonstram que essa estratégia melhora significativamente a forma das células da malha simplificada, gerando um menor erro de simplificação a longo prazo na maioria dos casos, o que pode ser explicado pela chance menor de inversões nas células hexaédricas.

Operação

Nosso algoritmo recebe um número alvo de células e opera em uma malha de reservatório que contém uma matriz quádrica e um número de atributos associados a cada vértice.

Esta malha é simplificada por uma série de colapsos de colunas. A próxima coluna a ser colapsada é escolhida com base no erro quádrico cometido, como em outros trabalhos (24, 58, 21). Os seguintes passos são efetuados a cada iteração do algoritmo:

- prepare aleatoriamente 8 colunas candidatas a colapso:
  - obtenha uma coluna aleatória;
  - identifique o tipo de colapso pela topologia local e calcule as novas posições e atributos dos vértices;
  - cheque se o colapso é válido; caso não seja, descarte a coluna como candidata;
  - calcule o custo de colapsar a coluna, considere a coluna uma candidata.
- dentre as 8 colunas candidatas, obtenha a coluna de menor custo, efetue o colapso da coluna;
- detecte e remova colunas em forma de *doublet* conforme descrito anteriormente;
- se a malha possuir um número de células menor ou igual ao número alvo, pare.

Parte dessa estratégia foi inspirada pelo trabalho de Wu e Kobbelt (58), que propõe escolher o melhor dentre um pequeno número de candidatos a colapso escolhidos aleatoriamente, ao invés de manter uma fila de prioridades atualizada com cada possível colapso, que atestamos ser muito custoso em nossos experimentos. Nossa implementação guarda em cada coluna as posições e atributos calculados para cada vértice, uma marcação de validade e o custo calculado. Esses dados são reutilizados sempre que possível e são invalidados sempre que necessário.

5.2 Hierarquia de Multi-resolução

Como o nosso algoritmo de simplificação de malhas de reservatório se baseia no colapso de colunas, esse trabalho propõe utilizar uma estrutura de multi-resolução 2D. Essa estrutura é uma versão 2D da estrutura *Adaptive TetraPuzzles*, apresentada por Cignoni et al. (18).

Assim como no trabalho de Erikson et al. (11), cada nó dessa estrutura possui uma simplificação da malha de todos os seus nós descendentes. A idéia é que cada nó possua uma quantidade de geometria aproximadamente igual, sendo pequena o suficiente para permitir uma variação progressiva de detalhe porém grande o suficiente para permitir malhas otimizadas e de tamanho adequado para as placas gráficas modernas. A Figura 5.10 ilustra a nossa hierarquia para malhas de reservatórios.

O primeiro passo para a geração da nossa estrutura é completar as colunas do modelo, conforme descrito na Seção 5.1.1. Em seguida, a caixa

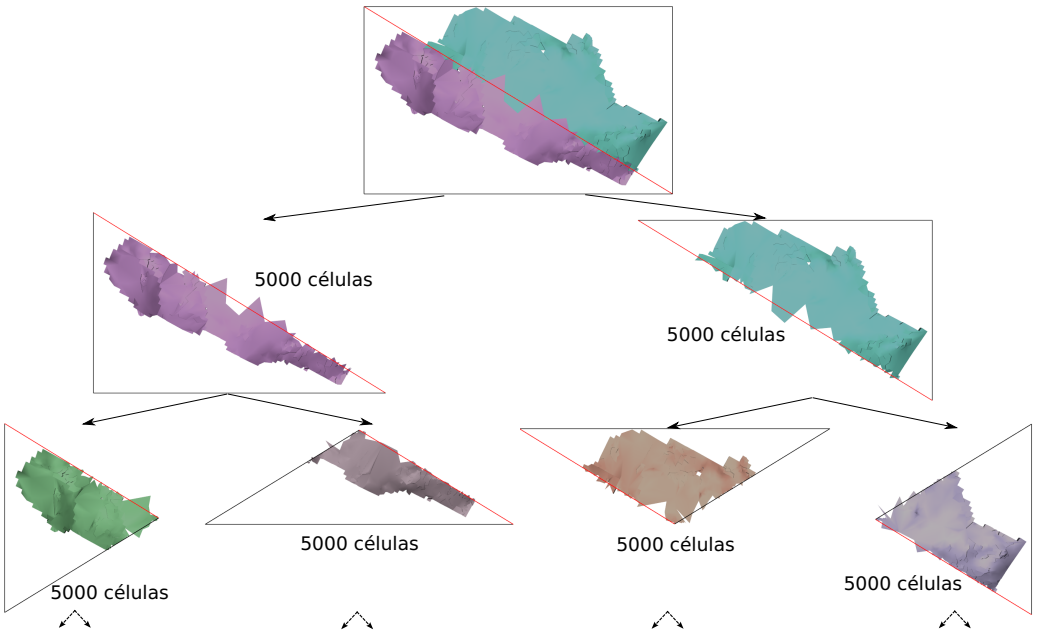


Figura 5.10: Hierarquia de multi-resolução utilizada. Hierarquia de regiões triangulares com a hierarquia de malhas simplificadas associada.

envolvente alinhada 2D do modelo (intervalos do domínio nos eixos x e y) é calculada e dividida em duas regiões triangulares. Cada célula do modelo original é associada a uma região se ao menos um dos seus vértices encontra-se dentro da região. Então, um processo recursivo se inicia a partir dessas duas regiões.

A cada passo de recursão é contabilizado o número de células associadas a uma dada região triangular. Caso esse número seja maior que um dado limite C_{max}, a região deve ser subdividida em duas regiões triangulares por biseção da maior aresta (arestas pintadas de vermelho na Figura 5.10). São calculadas as associações das células da região a cada região filha e o processo recursivo continua. Caso o número de células associadas à região seja aceitável, a região é feita folha da hierarquia. As células associadas são unidas e uma malha de hexaedros é formada. Calcula-se matrizes quádricas e atributos associados a cada vértice da malha conforme descrito nas seções anteriores. A malha é em seguida salva em disco com os dados para a sua simplificação posterior: uma marcação de atividade para cada célula, uma matriz quádrica e atributos associados a cada vértice. A malha final desses nós folha da hierarquia também é salva, onde apenas as células com o centróide (e não apenas qualquer vértice) dentro da região triangular são salvas.

Em seguida, um processo de simplificação de baixo para cima é iniciado na hierarquia, onde o objetivo é gerar representações cada vez mais simplificadas do modelo original. Esse procedimento explora o fato de que apenas

subdivisões conformes da hierarquia de regiões triangulares serão feitas em tempo de visualização. Como a subdivisão é sempre feita por biseção da maior aresta, uma subdivisão conforme (sem a presença de vértices T) é garantida se os triângulos que compartilham suas maiores arestas são ambos selecionados para a visualização ou ambos subdivididos. O par de triângulos que compartilham suas maiores arestas é denominado um *diamante*.

Dada essa assertiva de uso, o procedimento de simplificação de baixo para cima é feito de diamante em diamante: as malhas associadas aos triângulos internos de um dado diamante são calculadas primeiro se unindo as malhas dos filhos de cada triângulo do diamante. Em seguida essa malha é simplificada até que cada triângulo do diamante contenha C_{max} células. O casamento correto dos pedaços de malha é garantido pela observação dos três tipos de fronteira do diamante. O primeiro tipo de fronteira é chamado *fronteira interna*, dada pela maior aresta do diamante. É livre a manipulação de células que possuam interseção com essa fronteira, pois será garantido que ambas as malhas associadas aos triângulos do diamante serão ou renderizadas em conjunto ou substituídas pelas representações dos seus descendentes. O segundo tipo de fronteira é chamado *fronteira externa do modelo*, onde também é livre a manipulação de células intersectadas, pois não existe malha a ser casada do

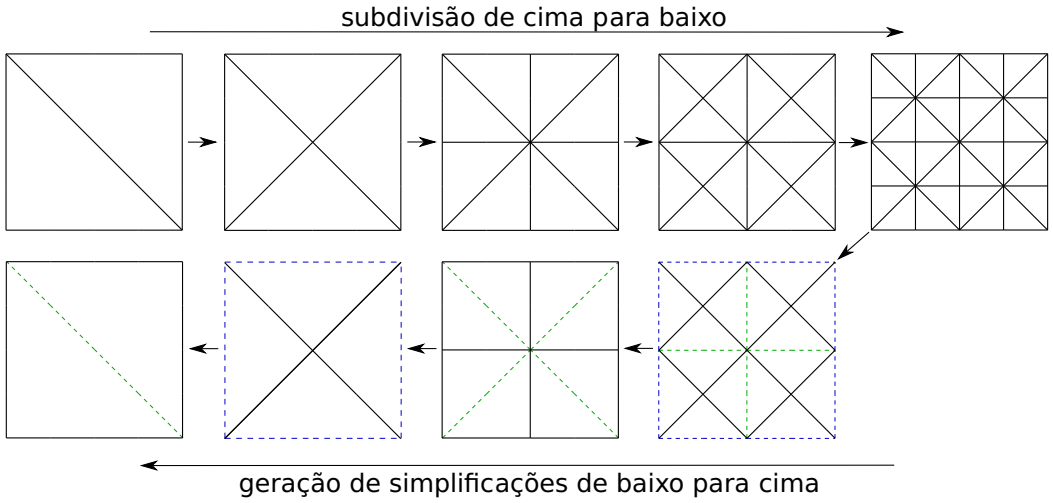


Figura 5.11: Ilustração do processo de subdivisão e de geração das malhas simplificadas. Na parte de cima é ilustrado cada nível do processo de subdivisão. Na parte de baixo é ilustrado cada nível do processo de geração de simplificações. As linhas cheias na parte de baixo ilustram as fronteiras externas dos diamantes. As linhas verdes em tracejado ilustram as fronteiras internas dos diamantes (maiores arestas de uma ou duas regiões triangulares). Já as linhas azuis em tracejado ilustram as fronteiras externas do modelo. A simplificação é livre nas fronteiras internas e nas fronteiras do modelo, e é proibida nas fronteiras externas do diamante.

outro lado da fronteira. O terceiro tipo de fronteira é chamado *fronteira externa do diamante*, onde devem ser travadas todas as células intersectadas.

Após completada a simplificação de um diamante, cada célula pertence à malha do triângulo que contém o seu centróide. A malha de hexaedros de cada triângulo é salva em seguida, novamente com os devidos atributos de células e de vértices.

Como cada diamante tem sua fronteira externa travada, é possível simplificar todos os diamantes de um dado nível hierárquico em paralelo, o que é uma grande vantagem se considerarmos a disponibilidade de processamento paralelo, como o uso de múltiplos núcleos de um computador ou até um agrupamento de PCs (*PC cluster*).

A Figura 5.11 ilustra o processo de subdivisão de cima para baixo e o processo posterior de geração de malhas simplificadas de baixo para cima. É possível notar na figura a forma pela qual as simplificações de diamantes em um mesmo nível são completamente independentes, possibilitando o uso de paralelismo.

5.3 Malhas para Visualização

Após preenchida a hierarquia de multi-resolução, as malhas de hexaedros geradas para cada nó devem ser preparadas para os algoritmos de visualização desejados.

Um dos algoritmos mais tradicionais é o chamado iso-contorno (*iso-contour*) das células, que renderiza as faces das células com uma cor representando um valor de um campo escalar. Esse algoritmo pertence a uma classe de algoritmos que utiliza as faces das células como suporte geométrico para a renderização. Conforme mencionado no Capítulo 3, esses algoritmos podem tirar proveito da extração das faces externas da malha de hexaedros. Essas faces são armazenadas em uma malha de polígonos, onde cada vértice contém uma posição, uma normal e seus outros atributos. Uma outra classe de algoritmos de visualização utiliza diretamente a malha de hexaedros, como por exemplo algoritmos de visualização volumétrica por traçado de raios (22).

Ambas as classes de algoritmos devem lidar com a natureza 2D do nosso algoritmo de simplificação, que reduz detalhe nas direções X e Y porém mantém a malha com um número possivelmente grande de finas camadas de células.

Esse trabalho propõe simplificar a malha de polígonos gerada para algoritmos que utilizam polígonos, o que será detalhado a seguir. Nossa proposta não simplifica as malhas de hexaedros resultantes da simplificação

na direção K, deixando o tratamento de algoritmos que utilizam diretamente esse tipo de malha como trabalho futuro.

5.3.1 Simplificação das Faces Laterais

No caso de algoritmos que utilizam malhas de polígonos, é possível visualizar as camadas do reservatório em conjunto ou em separado. Caso um conjunto de camadas seja visualizado sem separação, torna-se possível simplificar ainda mais as faces laterais da malha de polígonos. Isso é feito convertendo a malha de quadriláteros laterais em uma malha de triângulos e a simplificando até um erro quádrico máximo, que no caso da hierarquia utilizada pode ser facilmente calculado para cada malha associada a uma região triangular: basta avaliar o erro quádrico máximo dentre os seus vértices: $v^T Q v$, onde v é a posição do vértice e Q é a sua matriz quádrica associada.

Duas restrições devem ser impostas à simplificação para garantir a criação de malhas conformes (sem a presença de vértices T). Um vértice da malha de triângulos deve ser travado caso seja parte de uma face externa de topo ou de base. Também devem ser travados os vértices que se encontrem fora da região triangular associada, garantindo assim que as malhas casam na fronteira entre as regiões.

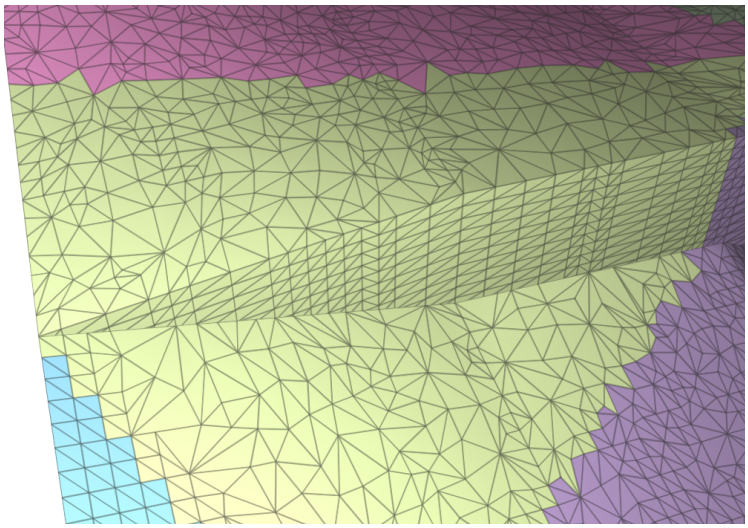
É possível reduzir a malha poligonal final significativamente dessa forma, especialmente quando lidamos com reservatórios com muitas camadas. Nossa implementação utiliza um simplificador simples de malhas de triângulos, com base na simplificação gulosa iterativa de colapsos de arestas (24). A Figura 5.12 ilustra a redução no número de polígonos da malha resultante da extração de faces externas laterais após a introdução deste processo de simplificação.

5.3.2 Organização dos Dados

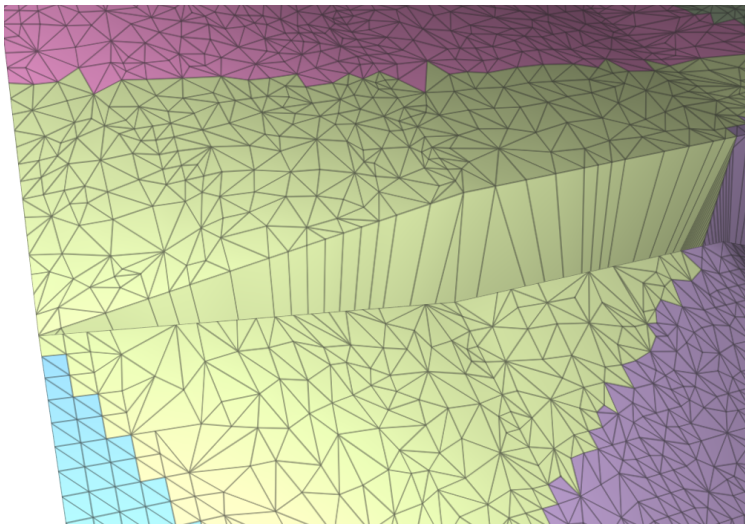
Como o objetivo do sistema proposto é prover visualização pelo uso de apenas um computador, assume-se que todos os dados de pedaços da malha estão armazenados em uma unidade de armazenamento secundária visível pelo motor de visualização.

Nosso visualizador permite o desenho do iso-contorno das células. Assim, as faces externas associadas a cada nó da hierarquia são extraídas, é feita a simplificação de faces laterais e em seguida a malha de polígonos final é armazenada.

Calcula-se as normais de cada vértice da forma usual, tirando médias das normais das faces que contém o dado vértice. É tomado o cuidado adicional de



5.12(a): Malha sem simplificação das faces laterais



5.12(b): Malha com simplificação das faces laterais

Figura 5.12: Simplificação da malha de polígonos gerada pela extração das faces externas laterais: (a) Malha sem simplificação (b) Malha com simplificação das faces externas laterais. As malhas associadas a cada região triangular foram desenhadas cada uma com uma cor distinta.

se duplicar vértices laterais nos cantos do modelo, que são utilizados por faces com normais muito diferentes. Cada vértice utilizará o armazenamento de:

- posição - 3 valores de ponto flutuante (12 bytes);
- um vetor normal, que é armazenado em 2 bytes utilizando métodos de quantização de normais unitárias;
- atributos adicionais.

As malhas das faces de topo, das faces de base e das faces laterais são armazenadas em separado. É feito o cálculo dos cones de normais (49) das faces

de topo e das faces de base. Isso permite que em tempo de visualização seja rápido descartar esses grupos de faces com base na sua orientação, conforme descrito no Capítulo 3.

Cada pedaço de malha é comprimido e salvo em um arquivo temporário. O conjunto de todos esses arquivos de malha é concatenado em um grande e único arquivo binário, denominado *arquivo binário da hierarquia*, e são armazenados as posições e tamanhos de cada malha em um arquivo texto. O arquivo binário da hierarquia será mapeado para a memória quando o motor de visualização for iniciado.

Nossa implementação utiliza a biblioteca LZO para a compressão sem perda de dados (40), permitindo uma descompressão rápida em tempo de visualização. Foi utilizado também o método de quantização de normais unitárias proposto por Baptista (6) para o armazenamento das normais das malhas.

5.4 Visualização Dependente do Observador

Esta seção descreve a parte do nosso sistema de visualização com multiresolução que lida com a renderização das malhas geradas no passo de construção da hierarquia. Após descrita a forma de selecionar as simplificações utilizadas na renderização de cada quadro, detalharemos a organização e operação do nosso motor de visualização.

5.4.1 Medição de Erro Projetado

A seleção de quais pedaços de malha serão renderizados, denominado um *corte* na hierarquia, é feito com base no cálculo de erros de simplificação projetados. O objetivo é utilizar malhas mais grosseiras onde seu erro de simplificação é menos perceptível.

O primeiro passo para estimar a percepção do erro de simplificação em uma dada malha é medir o tamanho do erro em coordenadas do modelo. Nós utilizamos o método proposto por Lindstrom (42) para converter erros quádricos em unidades de distância. Esse método inclui a área de cada face na matriz quádrica de seus vértices, fazendo com que o erro quádrico tenha dimensões de volume ao quadrado. A partir disso, uma aproximação do erro no espaço do modelo é dada pelo diâmetro de uma esfera com volume igual à raiz quadrada do erro quádrico. Um limite superior consistente para esse erro projetado na tela é medido como o tamanho aparente dessa esfera centrada

no ponto do volume envolvente da malha que está mais próximo do plano de projeção do volume de visão (*near plane*).

O trabalho original *Adaptive TetraPuzzles* (18) calcula dois volumes envolventes para cada pedaço de malha: um para uso em algoritmos de descarte por visibilidade e outro para o cálculo de erros projetados. Os volumes envolventes utilizados para estimar erro são feitos iguais para todos os tetraedros em um diamante, efetivamente englobando todas as malhas em um diamante, e calculados para envolver os volumes envolventes dos tetraedros filhos do diamante. O mesmo é feito para armazenar erros no espaço do modelo: é calculado o erro máximo dentre os tetraedros de um diamante, e esse valor é armazenado igualmente em todos esses tetraedros. O erro também é tornado monotônico quando se desce na hierarquia: o erro de um nó pai é dado pelo máximo entre o erro da sua malha e o erro armazenado nos seus nós filhos.

Em tempo de visualização, a seleção dos pedaços de malha utilizados é feita por uma simples visitação de cima para baixo nas árvores binárias. A cada nó, avalia-se o erro projetado, verificando se ele se encontra dentro da tolerância máxima de erro: se estiver, sua malha associada é utilizada, caso contrário, segue a visitação de seus nós filhos, efetivamente forçando a subdivisão do tetraedro por biseção de sua maior aresta. Dada a monotonicidade dos erros no espaço do modelo e dos volumes envolventes e o fato de que a projeção para a tela também é uma função monotônica, todos os tetraedros em um diamante terão o mesmo valor de erro projetado, e esse erro decrescerá quando se desce na hierarquia. Essas propriedades forçam que a mesma decisão seja tomada em nós diferentes de um mesmo diamante, por consequência forçando a utilização de uma subdivisão conforme da hierarquia, garantindo que os pedaços de malha utilizados vão sempre casar perfeitamente (18).

Diferentemente do trabalho de Cignoni et al. (18), propomos utilizar volumes envolventes e erros no espaço do modelo individuais para cada região triangular (equivalente a cada tetraedro no algoritmo original 3D). Em tempo de visualização serão garantidas as propriedades do erro projetado descritas acima.

Nosso algoritmo se baseia em uma propagação de erro projetado feita em uma visitação por largura na hierarquia. Em primeiro lugar, apenas as regiões triangulares dentro ou com interseção com o volume de visão precisam ser consideradas para a garantia de uma subdivisão correta da hierarquia. A visitação é feita utilizando uma fila, que é iniciada pela inserção das duas regiões triangulares raiz da hierarquia. Sempre que uma região triangular dentro do volume de visão for visitada:

– seu erro projetado é calculado caso isso ainda não tenha sido feito no

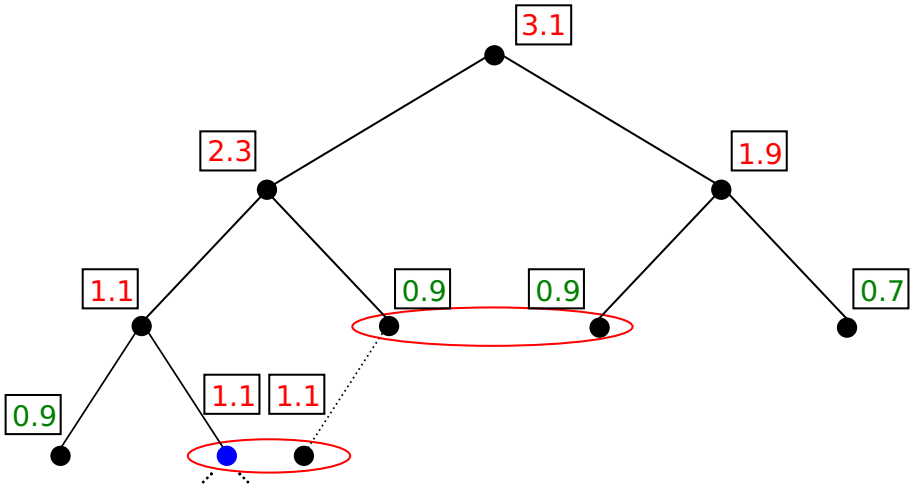
quadro corrente;

- seu erro projetado atual é comparado com o erro projetado do seu nó pai. Caso seja maior, esse erro é propagado para o nó pai, que é reinserido na fila de visitação;
- seu erro projetado atual também é comparado com o erro da outra região triangular em seu diamante. Caso seja maior, esse erro é propagado para essa outra região triangular, que é reinserida na fila de visitação;

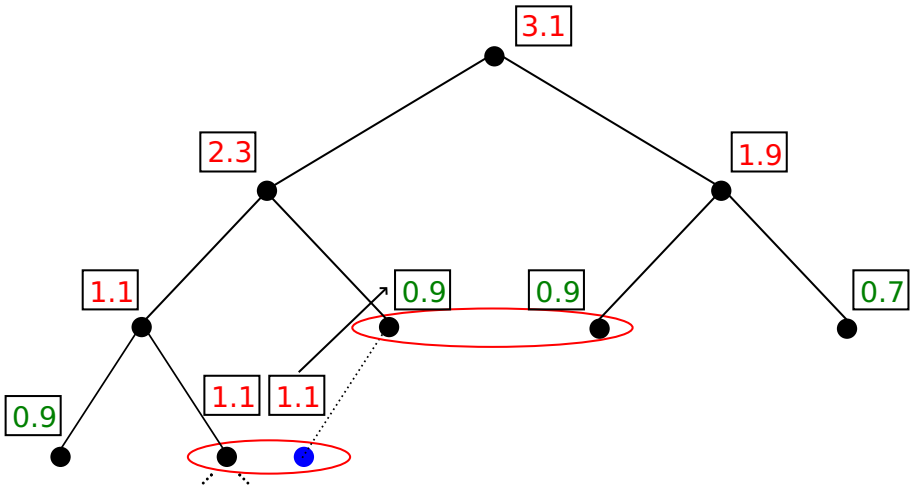
A Figura 5.13 traz uma ilustração do algoritmo acima descrito. Vamos assumir que o limite para o erro projetado seja de 1,0 *pixel*. Os nós da hierarquia são visitados por largura a partir da raiz. Quando o nó pintado de azul na Figura 5.13(a) é visitado, detecta-se que o outro nó em seu diamante não fora visitado. É propagado o erro projetado máximo no diamante (1,1) e esse outro nó é inserido na fila de visitação. Quando esse nó é visitado (pintado de azul na Figura 5.13(b)), é detectado que seu nó pai possui um erro projetado menor. O erro é propagado para o pai, que é reinserido na fila de visitação. Quando esse nó é novamente visitado (Figura 5.13(c)), seu erro (1,1) é propagado para o outro nó em seu diamante, que havia sido previamente aceito. Esse nó é reinserido na fila, e vai eventualmente ser dividido em dois em um próximo passo de visitação.

O algoritmo proposto vai sempre chegar a um fim, pois os erros projetados só crescem e só são propagados para cima ou para os lados na hierarquia. Dessa forma, as propriedades necessárias para um corte correto na hierarquia são garantidas, e é feita uma estimativa de erro projetado bem menos conservadora que a feita no algoritmo original. Os volumes envolventes podem ser individuais por nó, efetivamente fazendo uma estimativa inicial bem menos conservadora para o erro projetado, dado o uso desses volumes na estimativa desse erro. Os volumes envolventes também só devem englobar as suas próprias malhas, ao invés de englobar todas as malhas de todos os nós filhos de todos os nós de um diamante. Isso é especialmente importante, pois a junção de volumes envolventes é um problema conhecidamente difícil.

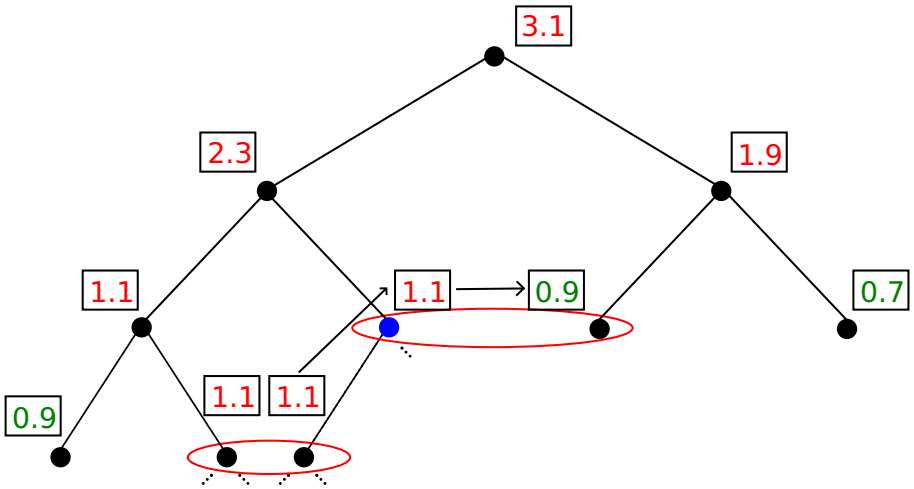
Esse algoritmo alternativo pode ser utilizado diretamente na renderização de hierarquias 3D geradas pelo trabalho original *Adaptive TetraPuzzles* (18).



5.13(a): Propagação para triângulo não visitado no diamante.



5.13(b): Propagação para triângulo pai.



5.13(c): Propagação para triângulo previamente aceito no mesmo diamante.

Figura 5.13: Ilustração do algoritmo de propagação de erro projetado proposto. Os nós sendo visitados estão pintados de azul. Os nós em um mesmo diamante estão envoltos por uma elipse vermelha.

5.4.2 Renderização em Tempo Real

Nosso motor de visualização é composto por duas linhas de execução: uma linha de execução para o carregamento de dados e outra para a renderização das malhas. A linha de execução de carregamento é responsável por ler as malhas da hierarquia de armazenamento secundário e disponibilizá-las para a linha de execução de renderização. Já a linha de execução de renderização deve selecionar as malhas carregadas que melhor representam o modelo, seguindo, é claro, as regras de subdivisão da hierarquia mencionadas anteriormente. As malhas selecionadas são então desenhadas utilizando as técnicas de aceleração consideradas.

Linha de Execução de Carregamento

Dados os parâmetros atuais de visualização, o objetivo da linha de execução de carregamento é possuir as melhores malhas para representar o modelo em memória principal quando a linha de execução de renderização calcular um corte na hierarquia. Para isso, calcula-se erros projetados da mesma forma que a linha de execução de renderização. Como o observador pode estar em movimento, há uma quantidade possivelmente grande de dados a serem carregados do disco a cada quadro, portanto é possível que nem todas as malhas consideradas ideais estejam carregadas em um dado momento.

Duas regras de carregamento de malhas da hierarquia são observadas. A primeira regra força que as malhas dos dois nós raiz da hierarquia estejam sempre carregadas, o que garante que pelo menos a representação mais grosseira do modelo estará disponível para a renderização. A segunda regra obriga que se é determinado que um nó deve ser carregado, então o seu nó pai também deve ser carregado. Ambas as regras nos permitem prover o melhor detalhe possível e não travar o *pipeline* gráfico enquanto as malhas são carregadas.

A linha de execução de carregamento somente carrega malhas cujo volume envolvente de visualização tem interseção com o volume de visão. Com isso, a memória principal somente armazenará pedaços de malha que possuem uma boa probabilidade de serem utilizados.

Ela também tenta predizer quais malhas serão utilizadas em um futuro próximo, dada a movimentação do observador. Isso é feito pelo uso de um volume de visão ligeiramente exagerado em relação ao utilizado na renderização e pela tentativa de se predizer a posição e orientação futura do observador. Essas estimativas são calculadas utilizando as velocidades lineares e angulares atuais, calculados a partir dos parâmetros de visualização de quadros anteriores.

Esta linha de execução é inicializada ao se mapear o arquivo binário da hierarquia para a memória, procedimento esse conhecido como *memory-mapping*. Calcula-se um volume de visão para a predição e propaga-se os erros projetados sempre que os parâmetros de visualização mudarem. Requisita-se o carregamento de todos os nós da hierarquia visitados durante a propagação de erros projetados e que estão dentro do volume de visão utilizado. O carregamento de dados é feito de forma assíncrona: informa-se o sistema operacional de que a região ocupada por uma malha no mapeamento do arquivo deve ser trazida para a memória principal.

Ao fim do laço principal da linha de execução de carregamento são detectadas quais malhas requisitadas já se encontram em memória principal. É feita então a descompressão dos dados e em seguida a malha é disponibilizada para a linha de execução de renderização. Também é detectado se o uso de memória principal passou de um determinado limite recomendado, sendo nesse caso liberadas as malhas que não tenham sido utilizadas ultimamente (*least recently used*).

Linha de Execução de Renderização

Dados o volume de visão da visualização e as malhas carregadas, a linha de execução de renderização deve a cada quadro calcular os erros projetados. Um teste adicional deve ser efetuado sempre que um dado nó da hierarquia tenha erro projetado maior que o limite: ele só pode ser subdividido se seus nós filhos estiverem carregados. Adicionalmente, os nós filhos do outro nó em seu diamante também devem estar carregados. Se a subdivisão de um nó for negada por conta disto, assume-se temporariamente que o nó possui um erro projetado *virtual* de 0 *pixels*. A mesma decisão será feita no outro nó do diamante, garantindo uma subdivisão correta.

Após decidir quais nós da hierarquia têm erro aceitável, conjunto esse chamado de *fronte* da hierarquia, as malhas selecionadas são renderizadas utilizando descarte por volume de visão, descarte por oclusão e descarte das faces de trás.

5.4.3 Garantindo uma taxa mínima de renderização

Como o nosso problema de renderização é majoritariamente limitado pelo estágio da geometria, é razoável assumir que, quanto mais primitivas sejam desenhadas num dado conjunto de quadros, menor será a taxa de renderização (*frame rate*). Entretanto, determinar precisamente quantas primitivas resul-

tarão numa dada taxa de renderização é muito difícil, senão inviável, dada a complexidade das arquiteturas das placas gráficas modernas.

No entanto, é possível implementar o que é chamado de um *escalonador reativo* (*reactive scheduler*) (32): com base nos tempos de renderização anteriores, é possível adjustar o número de primitivas no fronte da hierarquia, trazendo assim a taxa de renderização para mais próximo da taxa alvo.

Um dado quadro é renderizado utilizando um fronte com p primitivas dentro do volume de visualização[2]. Dada a coerência na renderização de quadros consecutivos, podemos assumir que se os últimos N quadros utilizaram em média p primitivas e levaram t segundos para serem renderizados, então é razoável ajustar o tamanho do fronte para conter $p' = \frac{pt'}{t}$ primitivas, onde t' é o tempo de renderização ideal por quadro.

Propõe-se então o seguinte algoritmo para garantir um número máximo de primitivas para o quadro sendo renderizado. O fronte inicial é composto pelos dois nós raizes da hierarquia. Em seguida aumenta-se o fronte atual a cada passo segundo os seguintes procedimentos:

- obtenha o nó do fronte com maior erro projetado E_{max};
- propague erros utilizando um limite igual a E_{max} menos uma pequena tolerância, o que é feito inserindo os nós que compõem o fronte em uma nova fila de propagação;
- caso o novo fronte possua mais primitivas que p', consolide o fronte do passo anterior como resultado final e pare.

A manutenção de uma taxa de renderização mínima é completada ao se informar a linha de execução de carregamento sobre o limite de erro projetado utilizado. O algoritmo proposto também corrige automaticamente o erro projetado considerado aceitável: se em um dado momento da visualização a taxa de renderizaçao for acima da taxa alvo, o limite de erro projetado diminuirá, automaticamente aumentando o detalhe geométrico, e vice-versa.

Dessa forma, o nosso motor de visualização consegue manter taxas de renderização interativas, utilizando o detalhe geométrico da melhor forma possível segundo os nossos objetivos de simplificação, como o respeito às bordas do modelo e a falhas geológicas.

[2]A consideração de outras técnicas de aceleração além de descarte por volume de visão pode ser feita, mas não é vital para o funcionamento da proposta.

5.5 Resultados Experimentais

O sistema proposto foi testado em um PC equipado com um processador Intel Core i7 2,67 GHz com 8 núcleos de processamento, 16 gigabytes de memória RAM e uma placa de vídeo NVIDIA Geforce 480 GTX com 1,5 gigabytes de memória de vídeo. Todos os componentes da solução foram implementados utilizando a linguagem C++, a API gráfica OpenGL e a biblioteca de gerência de linhas de execução *pthreads*.

Os modelos utilizados atualmente na indústria possuem uma quantidade de células muito inferior à dos modelos que estamos lidando, dados os tempos ainda grandes de simulação e a ausência de ferramentas de análise de resultados como a que estamos propondo. Por conta disso, foram gerados 6 modelos de teste a partir de dois modelos reais com diferentes discretizações, variando de 25 a 240 milhões de células. Os modelos A25, A100 e A217 foram gerados a partir de um modelo que possui um grande número de falhas geológicas e uma borda externa lateral (fronteira do conjunto de células ativas) com muitas quinas. Já os modelos B50, B100 e B240 foram gerados a partir de um modelo que possui uma distribuição irregular de células inativas, que requer a completação de muitas colunas, causando, no caso de colunas internas do nosso modelo, a consideração de faces externas na fronteira entre as células ativas e inativas. Os tamanhos dos modelos utilizados nos experimentos encontram-se na Tabela 5.1.

A Tabela 5.2 traz informações sobre os tempos de pré-processamento dos modelos de teste. O tempo de partição leva em consideração a leitura do modelo original do disco, a partição de suas células, o cálculo de atributos de vértice e o salvamento das malhas associadas a cada região folha da hierarquia. O tempo de simplificação leva em consideração todo o processo de preenchimento da hierarquia, desde a junção de sub-malhas, a simplificação da malha de

Nome do Modelo	Dimensões do *grid* topológico ($ni \times nj \times nk$)	Número de células ativas	Número de células consideradas na simplificação
A25	$594 \times 660 \times 122$	24 M	24 M
A100	$1188 \times 1320 \times 122$	96 M	96 M
A217	$1782 \times 1980 \times 122$	217 M	217 M
B50	$1826 \times 900 \times 92$	50 M	79 M
B100	$1826 \times 1080 \times 161$	104 M	165 M
B240	$1826 \times 1575 \times 253$	238 M	378 M

Tabela 5.1: Modelos utilizados nos experimentos.

Nome do Modelo	Número de células consideradas	Tempo gasto na partição	Tempo gasto na simplificação
A25	24 M	0,2 h	0,7 h
A100	96 M	1,3 h	2,7 h
A217	217 M	2,8 h	6,8 h
B50	79 M	0,8 h	2,3 h
B100	165 M	2,3 h	5,3 h
B240	378 M	4,6 h	10,5 h

Tabela 5.2: Tempo de pré-processamento dos modelos utilizados.

hexaedros, a simplificação das faces laterais externas, o salvamento das malhas de hexaedros da hierarquia e o salvamento das malhas para a visualização com iso-contorno. A simplificação foi feita com uma implementação que simplifica os diamantes da hierarquia em paralelo, utilizando 6 núcleos de processamento do computador utilizado nos testes. A simplificação de faces laterais também foi feita em paralelo.

Os testes de desempenho da visualização foram feitos pela medição de diversas estatísticas, como a taxa média de renderização de quadros, o erro projetado médio das malhas utilizadas e o uso de memória principal e de vídeo ao longo de dois caminhos de câmera pré-definidos: um para os modelos A25, A100, A217 e outro para os modelos B50, B100 e B240. O caminho de câmera para a primeira lista de modelos inicia de uma vista superior, faz uma aproximação ao modelo até $t = 20s$ e navega próximo ao modelo, chegando na sua borda em $t = 40s$. A partir desse ponto, o observador se afasta do modelo, voltando a enquadrá-lo a partir do ponto inicial em $t = 65s$. O caminho de câmera para a segunda lista de modelos também inicia de uma vista superior, fazendo uma pequena aproximação ao modelo até $t = 15s$, quando a vista se volta para um dos lados do modelo em $t = 20s$. O observador então se aproxima do modelo até $t = 30s$, quando ocorre uma nova rotação na orientação. A partir disso, é feita uma navegação próxima ao modelo até chegar à sua borda no final do caminho de câmera, em $t = 90s$. Todos os testes de renderização foram feitos utilizando uma tela com resolução de 1580×720 *pixels*.

5.5.1 Proposta de Cálculo de Erro Projetado

Foi feito um teste comparando a taxa de renderização alcançada utilizando duas estratégias para a estimativa de erro projetado e garantia de uma subdivisão correta da hierarquia:

– a proposta do trabalho original *Adaptive TetraPuzzles* (18), que garante

uma subdivisão correta da hierarquia em tempo de construção, dado o uso de erros no espaço do modelo e volumes envolventes iguais para todas as regiões triangulares em um diamante e monotonicamente decrescentes quando se desce na hierarquia;

– a proposta feita por este trabalho, que garante uma subdivisão correta da hierarquia em tempo de visualização, fazendo uso de erros no espaço do modelo e volumes envolventes individuais por região triangular e de um algoritmo de propagação do erro projetado, que garante as propriedades necessárias para o erro ao longo da hierarquia.

Este teste renderiza o modelo A100 com um limite fixo de erro projetado ao longo do caminho de câmera mencionado anteriormente. A Figura 5.14 contém a plotagem da taxa de quadros por segundo alcançada utilizando as duas propostas de cálculo de erro projetado. Como é possível observar, o sistema apresentou taxas de renderização significativamente superiores ao se garantir a subdivisão correta da hierarquia em tempo de visualização, conforme proposto por este trabalho.

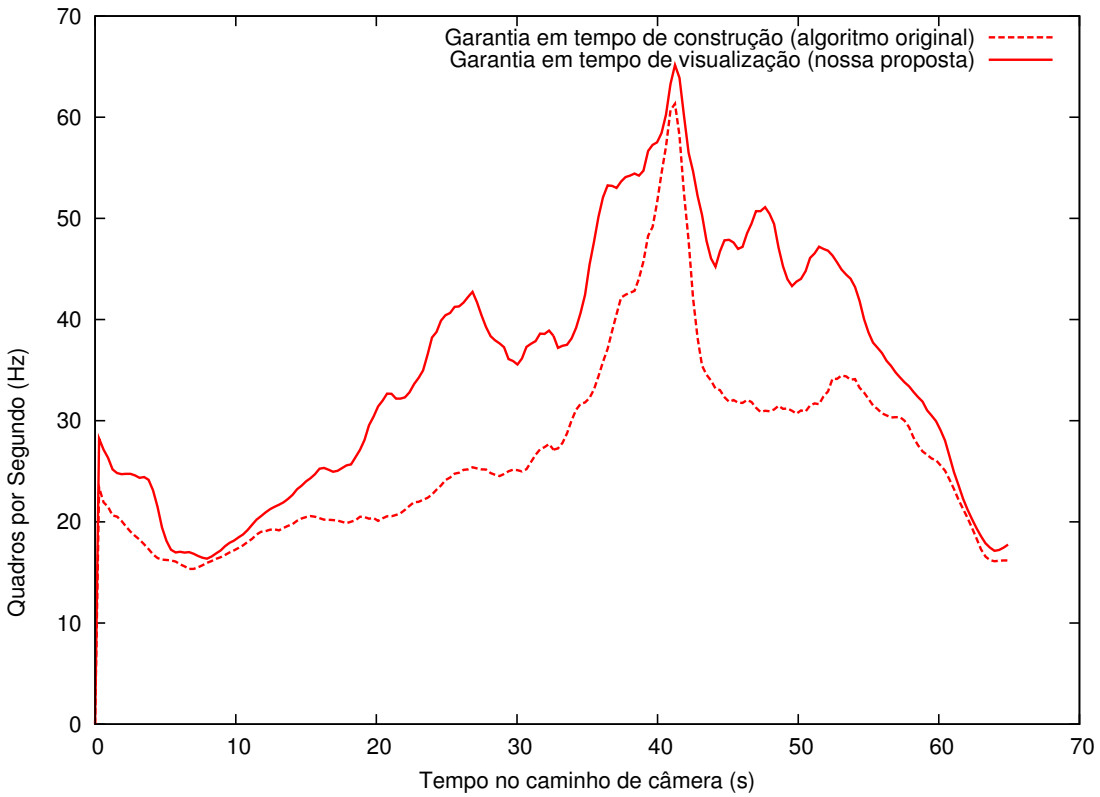


Figura 5.14: Comparação de taxas de renderização com diferentes métodos de garantia de uma subdivisão correta da hierarquia, modelo A100.

5.5.2 Proposta de Visualização em Tempo Real

Com o intuito de testar o nosso sistema de multi-resolução no que tange ao respeito aos limites de uso de memória principal e de uma taxa mínima de renderização, este teste traz a plotagem da taxa de quadros por segundo na renderização dos maiores modelos, A217 e B240, que possuem respectivamente 217 e 240 milhões de células. Em cada modelo foram feitos três testes:

- uso de um limite pequeno para o carregamento de malhas em memória principal (150 MB para o modelo A217 e 250 MB para o modelo B240);
- uso de um limite mais folgado para o carregamento de malhas (2 GB);
- uso do limite de 2 GB de RAM e da transferência em tempo real de malhas para a memória de vídeo, pelo uso da extensão ARB_vertex_buffer_object do OpenGL (*VBO*).

Todos os testes utilizam o algoritmo de garantia de taxa mínima de renderização proposto por este trabalho, sendo configurado o limite almejado de 30 quadros por segundo para ambos os modelos.

As Figuras 5.15 e 5.16 trazem as plotagens das taxas de renderização obtidas pelo sistema nos três testes para os modelos A217 e B240, respectivamente. Já as Figuras 5.17 e 5.18 trazem as plotagens do erro projetado médio nos mesmos testes. Dada a consideração de uma pequena margem, o desempenho do sistema cumpre o objetivo traçado ao longo de todo o caminho de câmera para ambos os modelos. O desempenho com as malhas na memória principal da CPU se mostrou bastante estável, sendo satisfatório mesmo com um limite pequeno para o carregamento de malhas. Em alguns casos obtivemos uma renderização com erro zero (modelo original) e taxa de renderização superior ao limite almejado, explicável pela grande aproximação aos modelos nesses trechos. Já o desempenho quando se utiliza VBOs, apesar de consideravelmente maior e com um erro consideravelmente menor, varia muito por conta dos tempos de transferência das malhas para a placa de vídeo. Em trabalhos futuros pretende-se experimentar um mecanismo de múltiplos níveis de memória como o descrito por Pinheiro e Velho (43), considerando as transferências entre o disco, a memória principal e a memória de vídeo. Outra opção possível seria dedicar um tempo fixo de cada quadro para a transferência de malhas da CPU para a GPU.

As Figuras 5.19 e 5.20 mostram o uso de memória principal pelas malhas do reservatório nas duas primeiras configurações de memória. O uso de memória se manteve abaixo dos limites configurados durante praticamente todos os caminhos de câmera. A Figura 5.21 traz fotos de tela do modelo B240

no nosso visualizador: a Figura 5.21(a) mostra o modelo simplificado utilizando 1 % das células do modelo original; a Figura 5.21(b) mostra a malha e as regiões triangulares utilizadas nesse modelo simplificado; e a Figura 5.21(c) mostra o modelo original visualizado do mesmo ponto de vista.

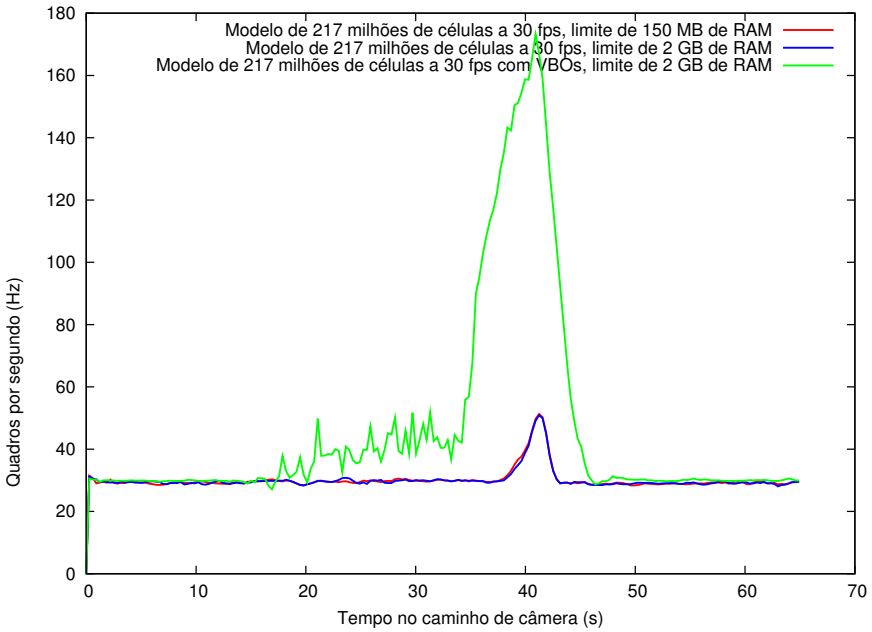


Figura 5.15: Garantia de uma taxa de renderização mínima de 30 quadros por segundo no modelo A217, que possui 217 milhões de células.

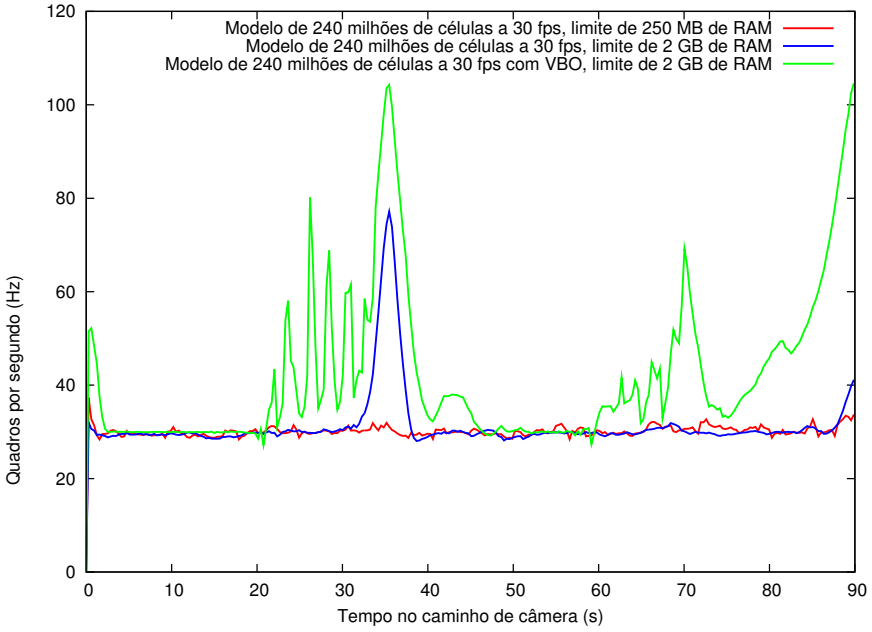


Figura 5.16: Garantia de uma taxa de renderização mínima de 30 quadros por segundo no modelo B240, que possui 240 milhões de células.

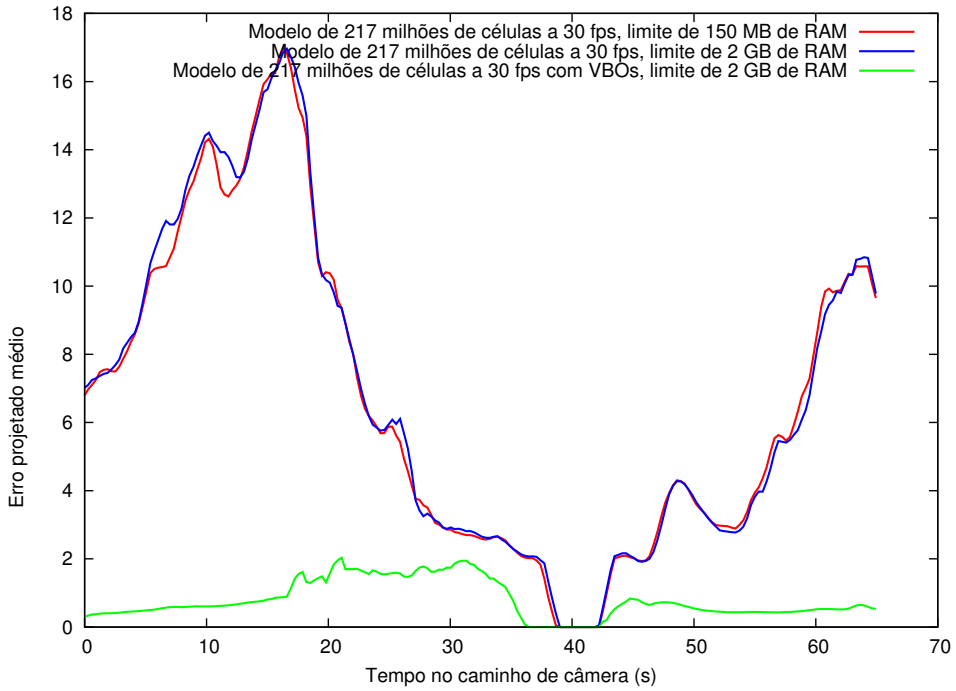


Figura 5.17: Erro projetado médio ao garantir uma taxa de renderização mínima de 30 quadros por segundo no modelo A217, que possui 217 milhões de células.

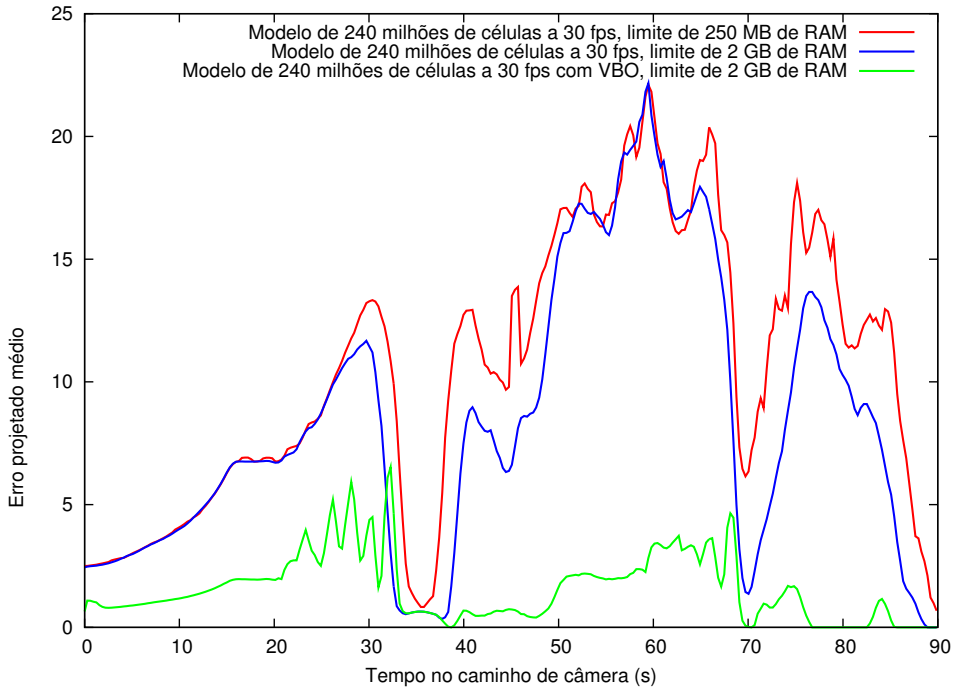


Figura 5.18: Erro projetado médio ao garantir uma taxa de renderização mínima de 30 quadros por segundo no modelo B240, que possui 240 milhões de células.

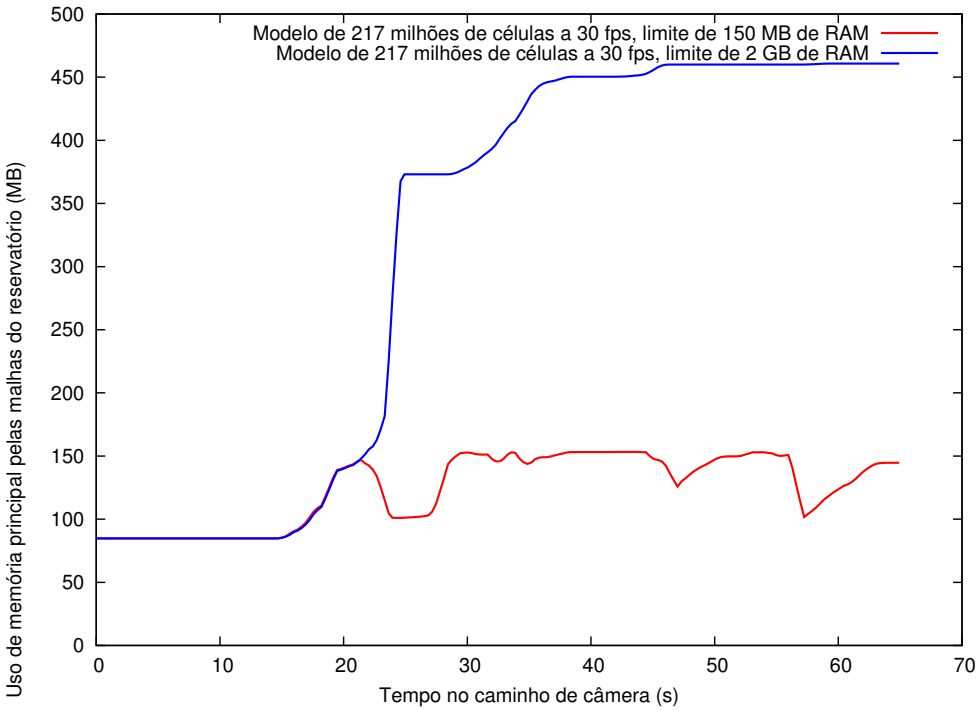


Figura 5.19: Uso de um limite para o uso de memória na visualização com taxa de renderização constante do modelo A217.

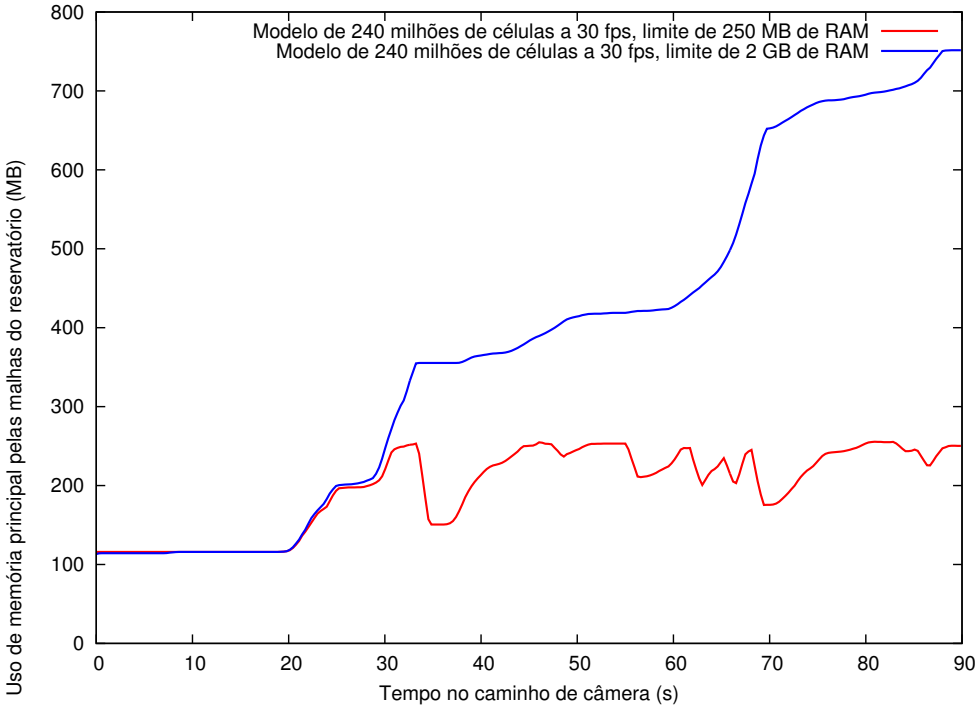
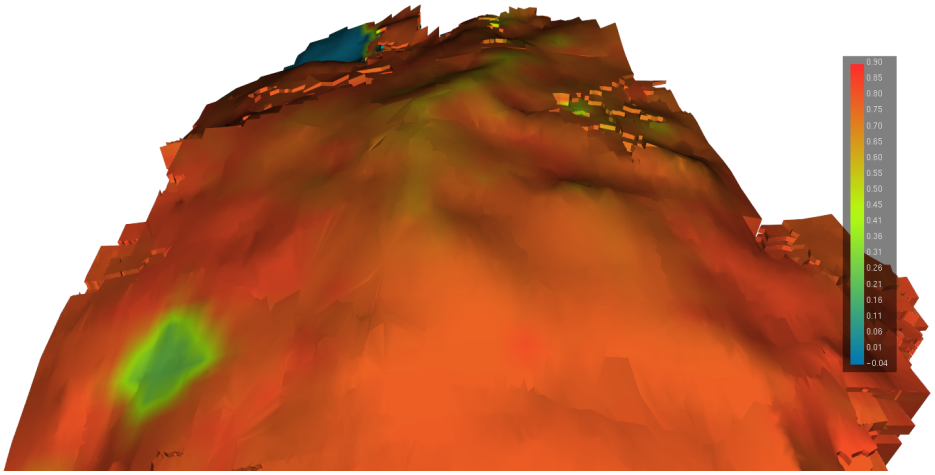
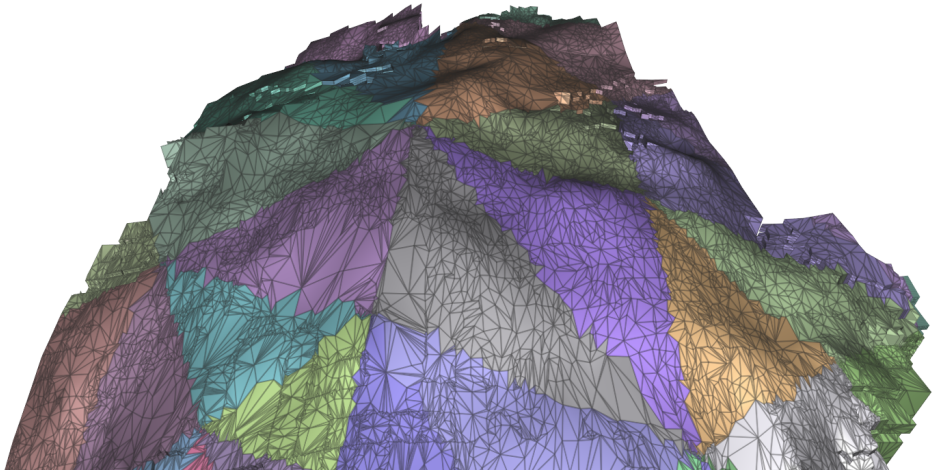


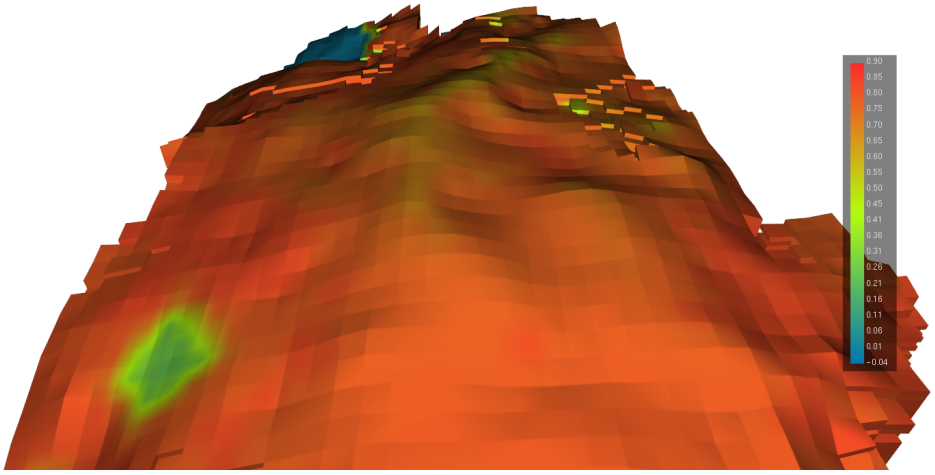
Figura 5.20: Uso de um limite para o uso de memória na visualização com taxa de renderização constante do modelo B240.



5.21(a): Modelo simplificado (1 % das células)



5.21(b): Malha e regiões triangulares utilizadas no modelo simplificado



5.21(c): Modelo original visualizado do mesmo ponto de vista

Figura 5.21: Fotos de tela do modelo B240 no nosso visualizador.

5.5.3
Proposta de Algoritmo de Simplificação

Com o propósito de avaliar as estratégias adotadas pela nossa proposta de algoritmo de simplificação de malhas de reservatórios naturais de petróleo, nós comparamos os resultados do algoritmo de simplificação seguindo quatro estratégias distintas.

A estratégia *MinError* escolhe a posição final para os novos vértices de uma coluna interna buscando a opção de menor erro em um *grid* paramétrico 4x4 dentro da coluna sendo colapsada (Figura 5.22). Em cada opção calcula-se as coordenadas x e y dos novos vértices em todas as faces de topo e de base da coluna seguindo os mesmos parâmetros, e a coordenada z é dada pela minimização de erro na direção z descrita na Seção 5.1.3. A opção que for livre de inversão de faces e que possuir o menor erro de simplificação é escolhida[3]. Esta estratégia não utiliza suavização nem no cálculo das posições dos novos vértices nem nos vértices da coluna que permanecem na malha.

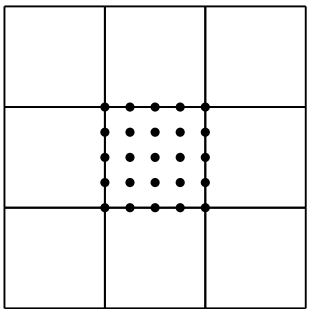


Figura 5.22: Vista 2D de cima do *grid* utilizado na estratégia *MinError*, que faz uma busca pela melhor posição final para os novos vértices de uma coluna interna.

A estratégia *Smooth* substitui a busca feita na estratégia *MinError* pela nossa opção de utilizar um processo local de suavização para definir as posições dos novos vértices e dos vértices que permanecem na malha. As duas direções de colapso para o primeiro e segundo operador de colapso são avaliadas nesta estratégia, sendo escolhida a de menor custo.

A estratégia *SmoothVal* parte da estratégia *Smooth* e inclui nossa opção por priorizar uma direção de colapso, o que foi feito levando em consideração a valência lateral dos vértices da coluna. Essa estratégia equivale à nossa proposta final.

[3]A posição de menor erro pode sem dúvida estar fora da coluna, porém seria muito custoso estender essa busca para as células adjacentes. O uso das matriz quádricas para a escolha das novas posições não é uma boa estratégia pois é necessária uma escolha coerente entre todas as faces de topo e de base da coluna.

Modelo	Número de células	Tempo total de simplificação			
		MinError	Smooth	SmoothVal	AllSmoothVal
A100	96 M	4,9 h	3,3 h	**3,1 h**	3,8 h
A217	217 M	11 h	7 h	**6,8 h**	8 h
B100	165 M	9,5 h	5,5 h	**5,3 h**	6,5 h
B240	378 M	18 h	11 h	**10 h**	12 h

Tabela 5.3: Comparação entre estratégias de simplificação: tempo de geração das simplificações dos modelos utilizados.

Já a estratégia *AllSmoothVal* inclui, a partir da estratégia *SmoothVal*, a suavização de todos os vértices de uma coluna sempre que ela é escolhida como candidata a colapso no nosso algoritmo iterativo, que foi descrito na Seção 5.1.3. A razão para experimentarmos essa estratégia é avaliar o benefício de uma suavização global da malha.

As quatro estratégias foram comparadas pela medição do tempo de pré-processamento gasto na geração das simplificações da hierarquia e pela medição da qualidade das simplificações geradas, tanto em termos de erro geométrico quanto em termos da forma das células hexaédricas geradas. A qualidade em termos de erro geométrico foi comparada pela medição do erro projetado médio na visualização com garantia de 30 quadros por segundo, feita ao longo de todo o caminho de câmera de cada modelo. Com o intuito de simplificar o teste, essa medição foi feita com todas as malhas já previamente carregadas e transferidas para a memória de vídeo do computador utilizado no teste. Já a qualidade em termos da forma dos elementos hexaédricos foi medida pela plotagem de um histograma dos jacobianos escalados das faces de topo das malhas, onde foram consideradas todas as malhas resultantes do algoritmo de simplificação ao longo da estrutura hierárquica. O jacobiano escalado é uma métrica que possui valor de 1,0 para quadriláteros retangulares, 0,0 para quadriláteros com três ou quatro vértices colineares e um valor negativo para quadriláteros côncavos, podendo chegar até -1,0. Idealmente todos os ângulos de uma face de topo (e de base) gerada pelo nosso algoritmo deveriam ser retos, correspondendo a um jacobiano escalado igual a 1,0.

As medições dos tempos de pré-processamento encontram-se na Tabela 5.3. Os resultados mostram que o tempo de simplificação é significativamente maior quando optamos por otimizar o erro em cada colapso (estratégia *MinError*). A estratégia *SmoothVal* se mostrou a de pré-processamento mais rápido, o que é explicado pelo descarte rápido de uma das direções de colapso no primeiro e segundo operador de colapso, sendo o primeiro o operador mais utilizado ao longo da simplificação. A estratégia *AllSmoothVal* adiciona um custo extra de suavização, que, como será observado nos gráficos a seguir,

compensa em termos de qualidade de malha, no entanto não compensa em termos de erro geométrico.

As Figuras 5.23, 5.24, 5.25, 5.26, 5.27, 5.28, 5.29 e 5.30 trazem a comparação do erro de simplificação e da qualidade dos elementos hexaédricos das malhas geradas por cada uma das quatro estratégias para os modelos A100, A217, B100 e B240, respectivamente. Conforme é possível observar em todos os gráficos de qualidade de malha, as estratégias *MinError*, *Smooth*, *SmoothVal* e *AllSmoothVal* possuem, nessa ordem, qualidades crescentes para a malha gerada, dada uma maior distribuição de primitivas próximo ao valor ideal para o jacobiano escalado de 1,0. Isso comprova que o uso de suavização e a consideração da valência dos vértices ao colapsar melhora a forma das células das malhas. Os gráficos de erro projetado médio ao longo do caminho de câmera mostram que a estratégia *SmoothVal*, que equivale à nossa proposta, obteve o menor erro geométrico projetado na renderização de todos os modelos, sendo superior às outras estratégias durante praticamente todo o tempo ao longo dos caminhos de câmera utilizados.

Os gráficos e tabelas apresentados justificam as decisões feitas ao projetar o algoritmo de simplificação proposto, apontando a nossa proposta como a estratégia que possui melhor custo-benefício, dados os tempos de pré-processamento e qualidade de simplificação apresentados.

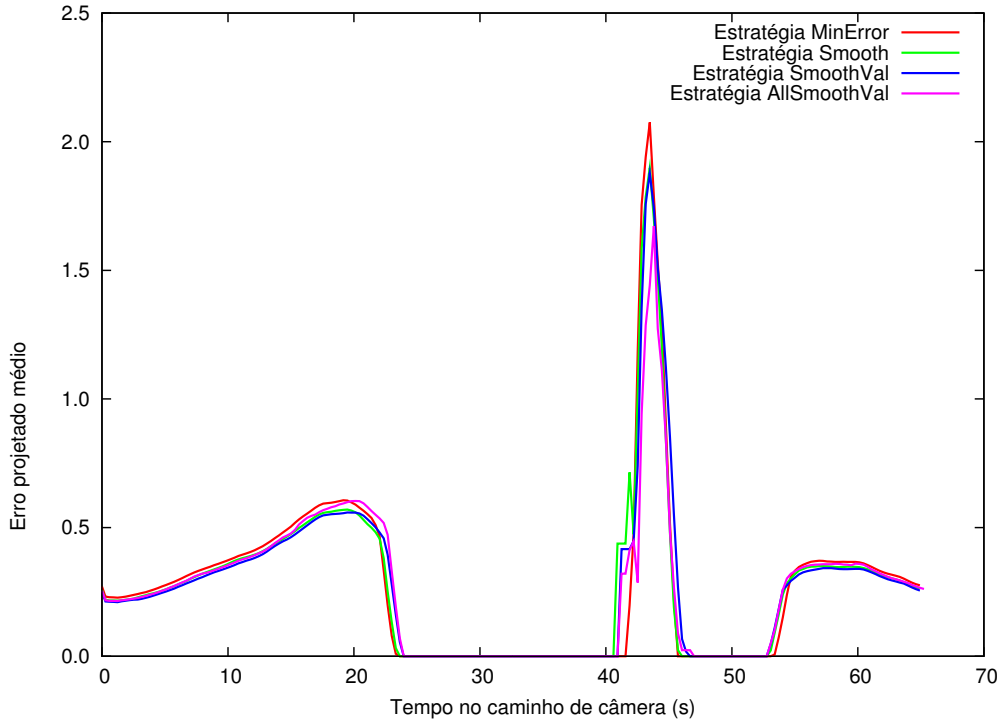


Figura 5.23: Comparação entre estratégias de simplificação no modelo A100: erro projetado médio ao longo do caminho de câmera.

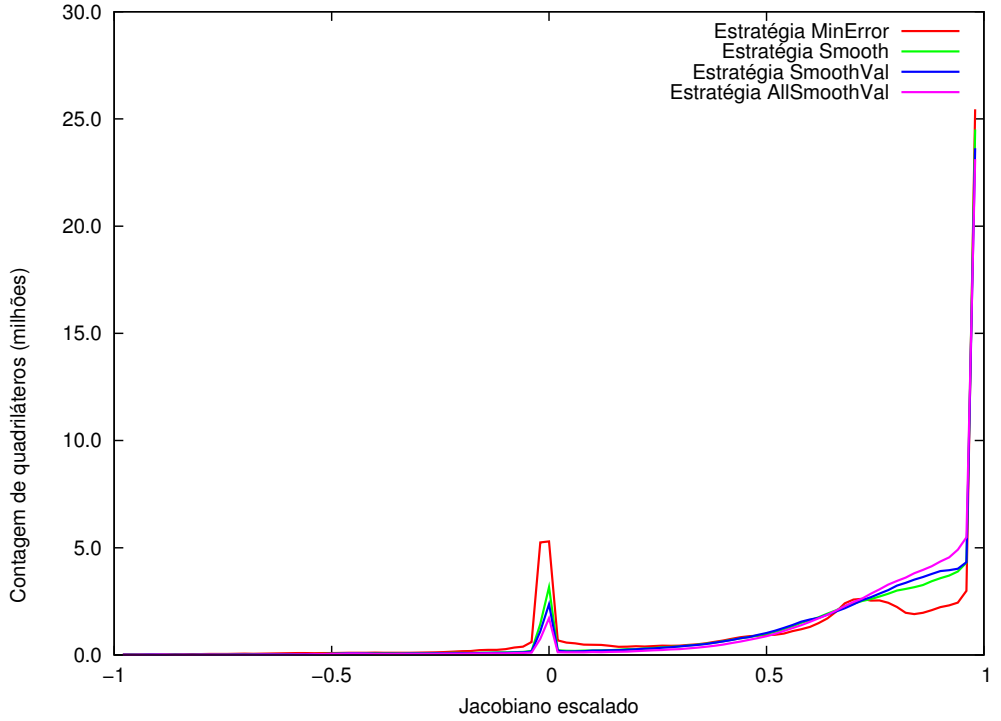


Figura 5.24: Comparação entre estratégias de simplificação no modelo A100: histograma da contagem de quadriláteros em cada nível de qualidade, medida pelo jacobiano escalado das faces de topo de todas as simplificações feitas na hierarquia.

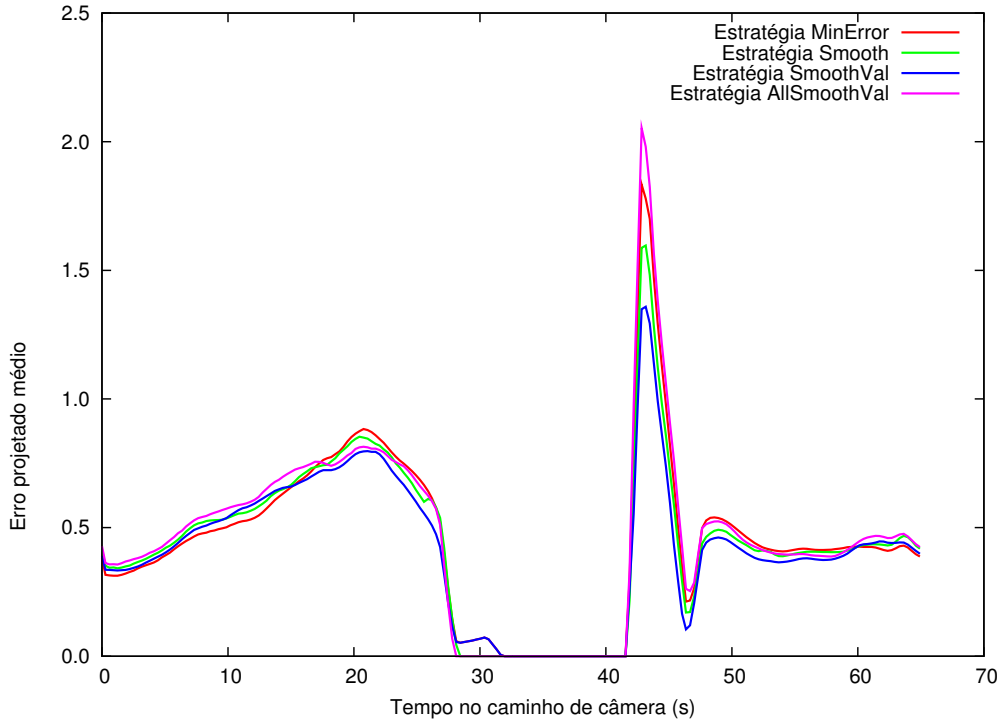


Figura 5.25: Comparação entre estratégias de simplificação no modelo A217: erro projetado médio ao longo do caminho de câmera.

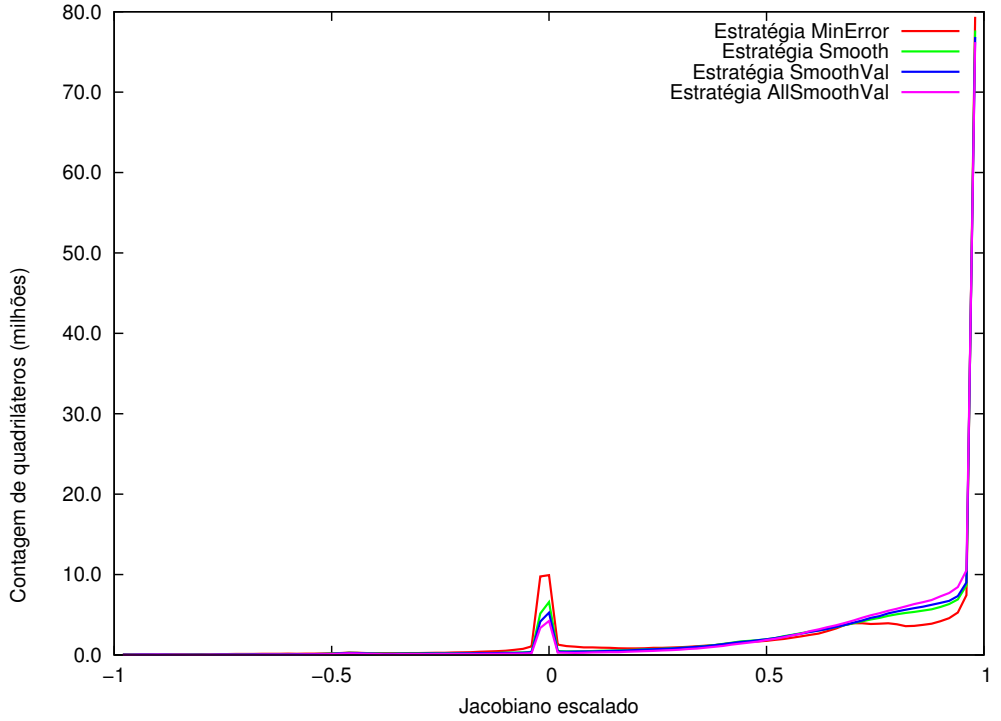


Figura 5.26: Comparação entre estratégias de simplificação no modelo A217: histograma da contagem de quadriláteros em cada nível de qualidade, medida pelo jacobiano escalado das faces de topo de todas as simplificações feitas na hierarquia.

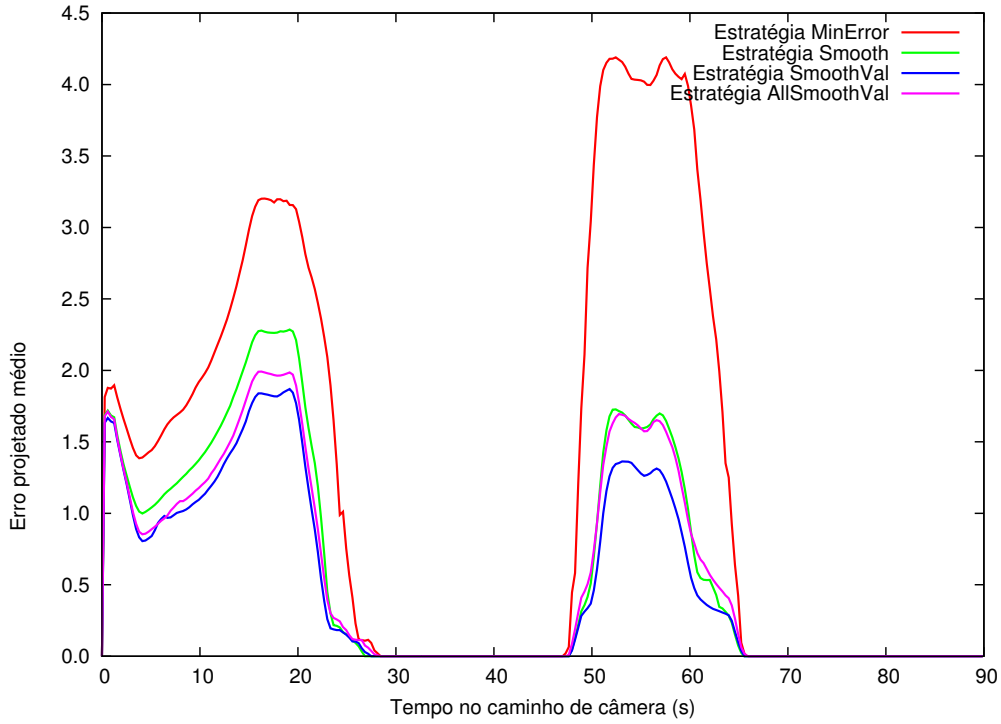


Figura 5.27: Comparação entre estratégias de simplificação no modelo B100: erro projetado médio ao longo do caminho de câmera.

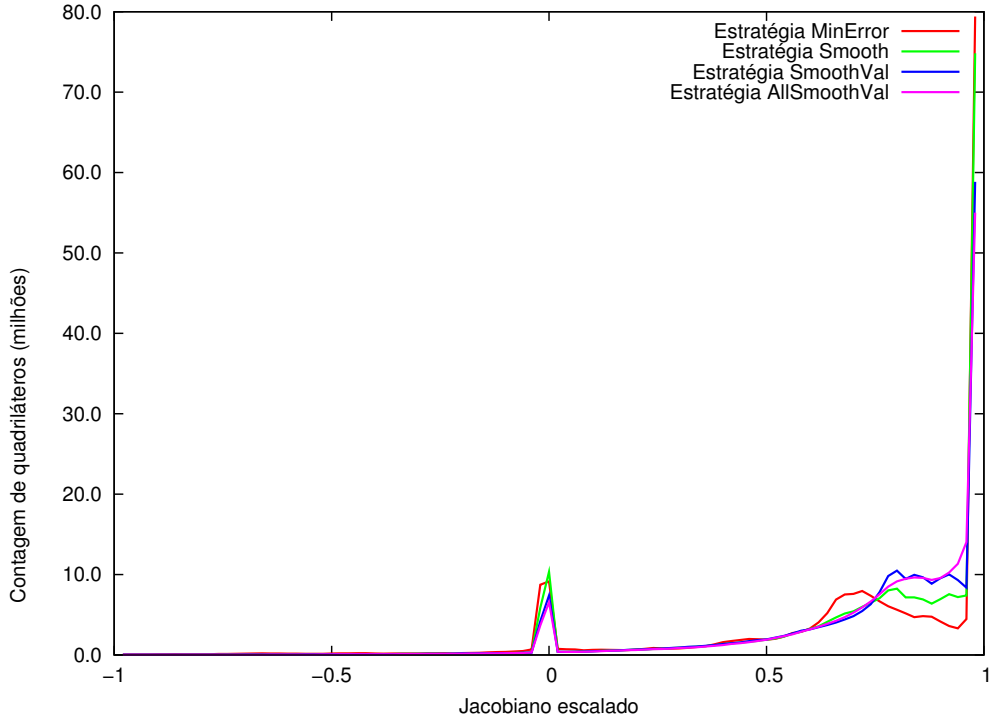


Figura 5.28: Comparação entre estratégias de simplificação no modelo B100: histograma da contagem de quadriláteros em cada nível de qualidade, medida pelo jacobiano escalado das faces de topo de todas as simplificações feitas na hierarquia.

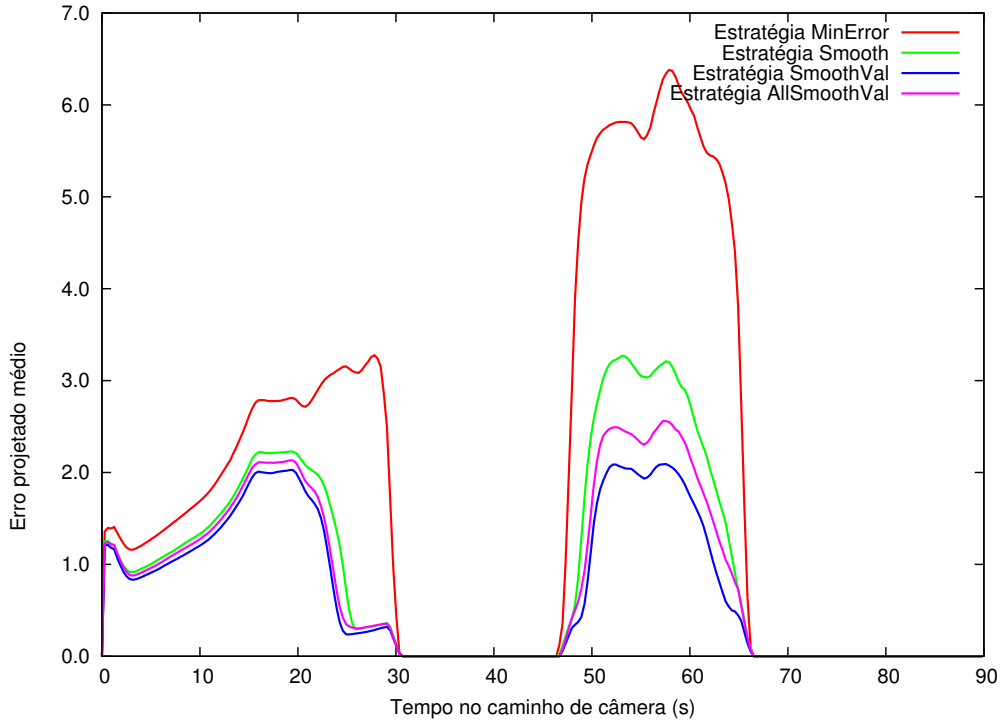


Figura 5.29: Comparação entre estratégias de simplificação no modelo B240: erro projetado médio ao longo do caminho de câmera.

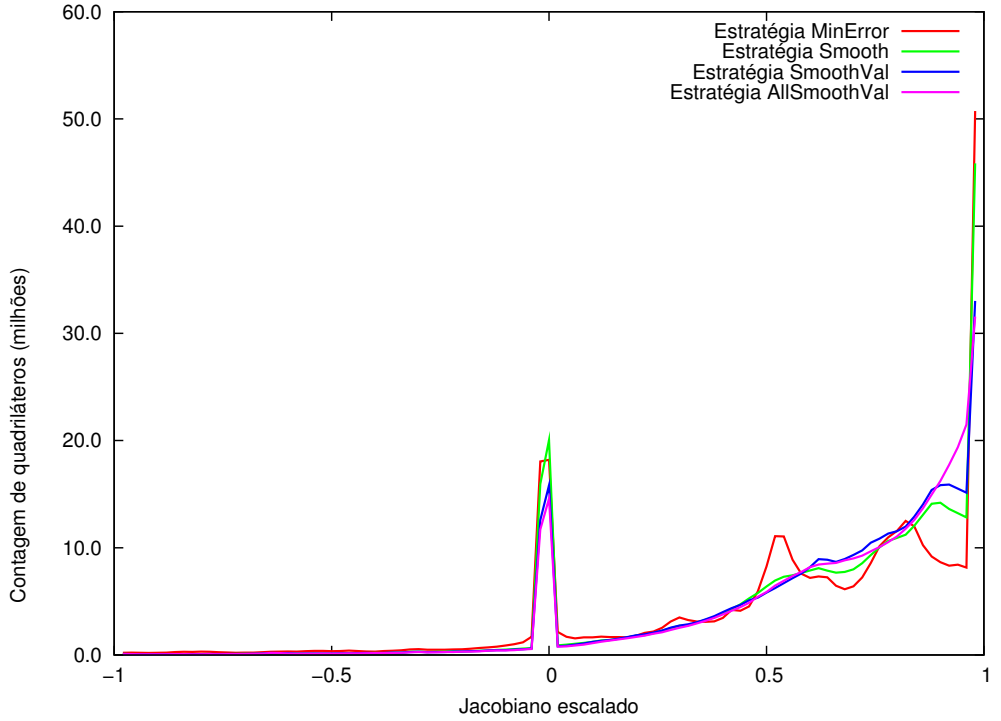


Figura 5.30: Comparação entre estratégias de simplificação no modelo B240: histograma da contagem de quadriláteros em cada nível de qualidade, medida pelo jacobiano escalado das faces de topo de todas as simplificações feitas na hierarquia.