Pontifícia Universidade Católica
do Rio de Janeiro

Samuel Bastos de Souza Junior

Extração de isosuperfícies de domos de sal em volumes binários massivos

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática, do Departamento de Informática da PUC-Rio.

Orientador: Prof. Waldemar Celes Filho

Rio de Janeiro
Setembro de 2020

Pontifícia Universidade Católica
do Rio de Janeiro

Samuel Bastos de Souza Junior

Extração de isosuperfícies de domos de sal em volumes binários massivos

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática da PUC-Rio. Aprovada pela Comissão Examinadora abaixo:

Prof. Waldemar Celes Filho
Orientador
Departamento de Informática – PUC-Rio

Dr. Jéferson Rômulo Pereira Coêlho
Departamento de Informática – PUC-Rio

Prof. Joaquim Bento Cavalcante Neto
UFC

Prof. Marcelo Gattass
Departamento de Informática – PUC-Rio

Rio de Janeiro, 25 de Setembro de 2020

Samuel Bastos de Souza Junior

Bacharel em Engenharia da Computação pela Pontfícia Universidade Católica do Rio de Janeiro (2018). É aluno do programa de pós-graduação do Departamento de Informática da PUC-Rio e desenvolvedor de software no Instituto TECGRAF.

Ficha Catalográfica

Bastos de Souza Junior, Samuel

Extração de isosuperfícies de domos de sal em volumes binários massivos / Samuel Bastos de Souza Junior; orientador: Waldemar Celes Filho. – 2020.

50 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2020.

Inclui bibliografia

1. Informática – Teses. 2. Extração de isosuperfícies. 3. Volumes sísmicos. 4. Campos de distância. 5. Estratégias out-of-core. I. Filho, Waldemar Celes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Primeiramente, gostaria de agradecer meu orientador, professor Waldemar Celes Filho. Desde a graduação, eu o tive como um exemplo de profissional e acadêmico a ser seguido. Por toda a atenção, apoio e paciência quando eu precisei, muito obrigado.

Gostaria também de agradecer ao meu amigo Jéferson Coelho, que mesmo com todas dificuldades e distância, fez questão de se mostrar presente e me ajudar, sempre que precisei. Obrigado, jovem.

À Gabrielle, que talvez tenha sido a pessoa que mais me apoiou e esteve por perto neste último período conturbado. Obrigado por tudo (e pela ajuda com os gráficos).

Agradeço meus companheiros de projeto no Instituto Tecgraf/PUC-Rio. Em especial gostaria de agradecer Carol e Leticia que, sem o apoio incondicional, talvez eu não tivesse conseguido ter chegado até aqui. Também agradeço o Instituto pelo apoio financeiro e educacional.

Por fim, mas não menos importante, agradeço à minha família e amigos, que foram os que viram meu esforço mais de perto e compartilharam das minhas dificuldades, me dando calma quando mais precisava.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamentode Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

Resumo

Bastos de Souza Junior, Samuel; Filho, Waldemar Celes. **Extração de isosuperfícies de domos de sal em volumes binários massivos**. Rio de Janeiro, 2020. 50p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Ao extrair isosuperfícies de dados volumétricos massivos, em geral a superfície de saída é densa, podendo demandar muita memória para seu processamento. Além disso, dependendo do método de extração utilizado, pode-se também obter um resultado contendo diversos problemas geométricos e topológicos. Neste estudo, experimentamos combinações de diferentes métodos de extração de isosuperfícies juntamente com estratégias *out-of-core* que permitem uso inteligente do recurso computacional para sintetizar aproximações poligonais dessas superfícies, preservando a topologia original segmentada. O método implementado foi testado em um volume sísmico real para extração da superfície de domo de sal.

Palavras-chave

Extração de isosuperfícies; Volumes sísmicos; Campos de distância; Estratégias out-of-core.

Abstract

Bastos de Souza Junior, Samuel; Filho, Waldemar Celes (Advisor). **Isosurface extraction of massive salt dome binary volume data**. Rio de Janeiro, 2020. 50p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

When extracting isosurfaces from massive volumetric datasets, in general, the output surface is dense, and may require a lot of memory for processing. In addition to this, depending on the extraction method used, the result can also include several geometric and topological problems. In this study, we experimented combinations of different isosurface extraction methods along out-of-core strategies to generate polygonal approximations to these surfaces, preserving the original topology segmented in the volumetric dataset. The implemented method was tested in a real seismic volume dataset for the salt dome extraction.

Keywords

Isosurface extraction; Seismic volumetric datasets; Distance fields; Out-of-core strategies.

Sumário

Lista de figuras

Lista de tabelas

Lista de algoritmos

I choose to fight this dream and drift into my lost reality.

Mercenary, *Lost Reality - The Hours that Remain.*

1
Introdução

1.1
Motivação

Este estudo surgiu de uma demanda da indústria de petróleo, onde deseja-se criar uma representação de um dado específico de domo de sal. Este dado é originado de um desafio *online* proposto pela TGS na plataforma Kaggle. Este desafio se trata da segmentação de um dado volumétrico de um domo de sal, identificando o que pertence e é interno ao domo, e o que é exterior.

Embora dados volumétricos sejam muito bem explorados na literatura, dados volumétricos criados a partir da aquisição geofísica de domos de sal não são muito comuns. Além de poderem apresentar uma topologia complexa, esses dados também são grandes o suficiente para que a memória principal de muitas máquinas não seja capaz de carregá-los inteiramente.

Com base nisso, vimos necessidade de estudar a literatura em busca de soluções. Baseado em trabalhos anteriores, como o método desenvolvido por [Lobello et al., 2014], propomos uma representação simples e ao mesmo tempo fiel à topologia do domo de sal. Objetivamos também o desenvolvimento de um procedimento que não seja muito custoso em termos de consumo de memória e de tempo.

1.2
Aquisição de dados volumétricos sísmicos

Dados volumétricos são importantes e podem ser originados de diversos domínios, mais reconhecidamente usados para criação de imagens médicas.

Esses dados costumam ser adquiridos usando técnicas como CT (*computerized tomography*), um processo de *scan* físico computadorizado utilizando raio-x. Neste método, raios-x são emitidos em direção ao corpo do paciente, e suas intensidades são capturadas e salvas do outro lado. Outro método bastante conhecido é o MRI (*magnetic resonance imaging*), baseado em ressonância magnética que permite identificar diferentes materiais no volume.

Em um contexto geofísico, essa aquisição é um pouco diferente. Estamos interessados neste trabalho no processamento de dados originados de aquisição sísmica. Explicando de uma forma simples, esses dados são obtidos através da propagação de ondas mecânicas no mar, em direção à área de interesse. Essas ondas atravessam o solo, onde parte delas são refletidas de volta para superfície e recebidas por receptores, que registram o tempo de chegada dessas ondas. Dependendo do tipo de formação rochosa essas ondas atravessam, altera-se o tempo de chegada. Esse dado bruto é então processado e organizado em uma grade tridimensional ou bidimensional, onde cada um dos *voxels* ou *pixels* possuem um valor de amplitude associado.

1.3 Estrutura da dissertação

No Capítulo 2, apresentaremos os trabalhos que estão intrinsecamente relacionados com o sistema desenvolvido, tendo como foco os principais métodos de extração de isosuperfície. No Capítulo 3, apresentamos os principais conceitos necessários para entender o sistema desenvolvido. O Capítulo 4 apresenta detalhadamente o sistema, suas etapas e particularidades. No Capítulo 5, apresentamos alguns resultados obtidos usando o sistema construído. Por último, apresentamos no Capítulo 6 nossa conclusão sobre o método desenvolvido e propomos alguns trabalhos futuros.

2
Trabalhos relacionados

Neste capítulo apresentamos os trabalhos que se relacionam com o sistema desenvolvido. A pesquisa surgiu a partir do estudo inicial do artigo [Lobello et al., 2014], onde buscávamos uma solução para extração da isosuperfície de interesse do dado volumétrico. [Lobello et al., 2014] combinam uma estratégia *out-of-core* com uma estratégia de extração de isosuperfície baseada em um algoritmo de *dual contouring* [Nielson, 2004].

Nosso método exige como entrada um volume previamente segmentado por alguma ferramenta ou algoritmo de segmentação. Existem diversos métodos e abordagens na literatura para a segmentação em dados volumétricos, alguns destes desenvolvidos especificamente para domos de sal. [Aqrawi et al., 2011] apresentam uma abordagem de segmentação baseada em processamento de imagem e no uso do operador de Sobel 3D, que é bastante utilizado também na detecção de arestas em imagens 2D. Existem também métodos que encaram o problema como um caso de uso para *machine learning* e utilizam redes neurais para diferenciar valores internos e externos à isosuperfície em volumes, como [Shi et al., 2018], [Shi et al., 2019] e [Di and AlRegib, 2017].

Para entender melhor a natureza da solução, revisamos alguns dos principais métodos de extração de isosuperfície. O *Marching Cubes* [Lorensen et al., 1987] é o mais conhecido. Ele divide o dado volumétrico em uma grade regular de cubos, que chamaremos de células, e cria um pedaço de superfície poligonal para cada célula em relação ao isovalor de interesse e com base nos valores que possuem em seus vértices e uma *look-up table* pré-computada. O *Marching Cubes* cria uma superfície *manifold*, mas não é capaz de representar *sharp edges* e *corners*. Métodos como o *Extended Marching Cubes* [Kobbelt et al., 2001], foram criados para melhorar a qualidade da superfície de saída, representando estes *sharp edges* e *corners* utilizando informações adicionais do volume, como valores de normais, para posicionar os vértices de maneira mais fiel à geometria.

[Ju et al., 2002] dividem também o volume em uma grade de cubos, criam um vértice representativo para cada cubo e a superfície de saída conectando esses vértices (Figura 2.1). Diferente de [Lorensen et al., 1987], esse trabalho é

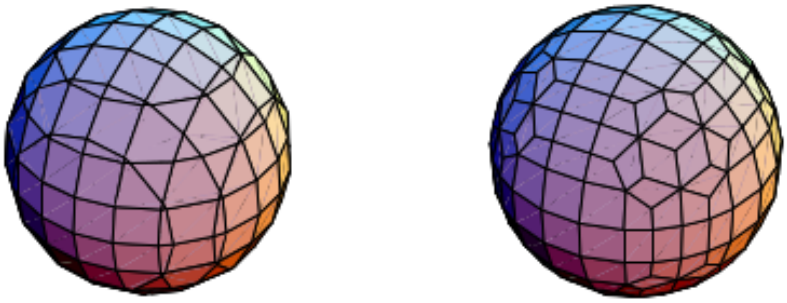

Figura 2.1: Uma esfera extraída a partir do algoritmo *Marching Cubes* (esquerda) e do algoritmo *Dual Contouring* (direita). Fonte: [Ju et al., 2002]

capaz de representar melhor a topologia, reproduzindo *sharp edges* e *corners* que o *Marching Cubes* não consegue, mas não é garantido de construir uma superfície *manifold*, e também pode apresentar polígonos que se interceptam. Além disso, não se limita a uma divisão regular do volume e pode ser combinado com estruturas hierárquicas, como *octrees*, para apresentar um resultado em multi-resolução.

[Ju et al., 2006] apresentam um passo complementar do método dos próprios autores em [Ju et al., 2002], que permite eliminar os casos de interseção de polígonos fazendo algumas verificações geométricas das conexões entre os vértices duais e suas respectivas células.

Com o *Dual Marching Cubes* [Nielson, 2004], foi superada a limitação de metódos duais poderem criar apenas um vértice representativo por célula, permitindo criar até quatro vértices distintos. A superfície é extraída examinando as arestas de cada célula e consultando uma tabela pré-computada que relaciona vértices duais com arestas. Os vértices são posicionados de acordo com quais de suas arestas bipolares são selecionadas pela tabela. [Nielson, 2004] também não garante uma superfície *manifold* e é aplicável apenas sobre uma grade regular. Embora não garanta a construção de uma superfície *manifold*, um simples tratamento permite contornar este problema.

[Schaefer et al., 2007] apresentam uma extensão dos trabalhos de [Ju et al., 2002] e [Nielson, 2004] onde, ao usar um simples critério topólogico de *cluster* de vértices duais, permite criar uma superfície sem vértices ou arestas não *manifolds*. Embora a superfície de saída ainda seja *manifold*, podem existir polígonos que se interceptam.

3
Conceitos

Este capítulo é dedicado a resumir os principais conceitos abordados nos próximos capítulos. Toda a terminologia usada é apresentada neste capítulo. Porém, nem todas as definições vão apresentar um formalismo matemático, apenas quando necessário para entendimento do conceito.

3.1
Campos escalares e isosuperfícies

Para entender o que é uma isosuperfície, é preciso primeiro entender o que é um campo escalar e suas características. Um campo escalar, segundo [Wenger, 2013], é uma função ϕ que atribui um valor escalar para cada ponto em $\mathbb{R}^d$, onde d denota a dimensão do campo escalar. Dado um campo escalar $\phi : \mathbb{R}^d \rightarrow \mathbb{R}$ e uma constante $\sigma \in \mathbb{R}$, existe um conjunto $\{x : \phi(x) = \sigma\}$, chamado conjunto de nível, ou $\phi^{-1}(\sigma)$.

Se ϕ é uma função contínua, o conjunto de nível separa $\mathbb{R}^d$ em dois conjuntos diferentes de pontos, os que possuem valor maior que σ, e os que possuem valor menor que σ. Dependendo da dimensão a qual o campo escalar pertence, o conjunto de nível recebe um nome diferente e, para $d = 3$, que é a dimensão interessada neste trabalho, recebe o nome de superfície implícita, ou isosuperfície. O valor σ recebe um nome especial também, chamado isovalor. Neste trabalho, chamaremos a aproximação poligonal do conjunto de nível de isosuperfície.

Essa aproximação do conjunto de nível, ou isosuperfície, pode ser calculada de muitas maneiras diferentes. A estratégia mais antiga se baseia em particionar o dado volumétrico em fatias 2D e construir o isocontorno (nome dado ao conjunto de nível em duas dimensões) de cada fatia. É possível obter a isosuperfície conectando esses isocontornos.

A estratégia mais conhecida, [Lorensen et al., 1987], particiona o dado volumétrico em cubos em relação ao isovalor e cria uma parte da superfície para cada uma desses cubos checando uma tabela pré-computada de 256 entradas diferentes, que são permutações ou espelhamentos dos possíveis valores dos vértices de um cubo.

A solução implementada neste trabalho segue a estratégia denominada *dual contouring*. Métodos baseados em *dual contouring* também particionam o volume em cubos, que chamaremos de células. Cada célula é representada por um [Ju et al., 2002] ou mais vértices [Nielson, 2004], e esses vértices são conectados a vértices de cubos adjacentes para compor a isosuperfície.

A vantagem de *dual contouring* é a facilidade de utilizá-lo juntamente como técnicas de multi-resolução. A maior desvantagem é exigir um tratamento adequado para que a isosuperfície extraída seja *manifold* e sem auto-interseção.

3.2 Dados volumétricos

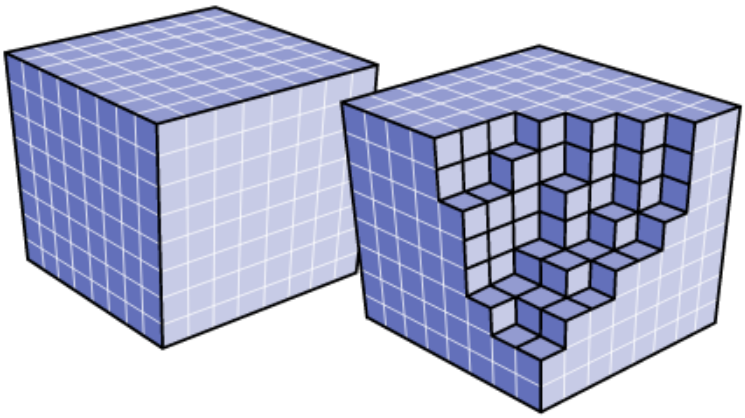

Figura 3.1: Ilustração de um volume como uma grade regularmente dividida. Cada cubo desta divisão representa um *voxel*. Fonte: [Engel et al., 2004]

Um volume, de acordo com [Engel et al., 2004], é um campo escalar 3D contínuo, pela definição dada anteriormente de campo escalar. Na prática, um campo volumétrico é uma discretização em grade deste campo contínuo. A forma mais fácil de explicar o que é um volume é traçando um paralelo com uma imagem 2D. Uma imagem 2D é uma grade bidimensional de elementos, chamados *pixels* (abreviação de *picture elements*).

Um dado volumétrico pode ser visto da mesma forma, porém em três dimensões. Um volume pode ser abstraído como uma pilha de imagens 2D. Ao invés de *pixels*, o nome dado a esses elementos passa a ser *voxels* (abreviação de *volume elements*). Existe mais de uma definição de *voxel*, neste trabalho vamos entender *voxel* como elementos infinitesimais de uma grade regularmente dividida que possuem um valor associado (Figura 3.1).

3.3 Campo de distância e Transformadas de distância

Um campo de distância é um campo escalar calculado a partir de um conjunto de pontos Σ e uma função de distância, que determina a menor distância entre um ponto p e outro ponto qualquer do conjunto:

$$dist_{\Sigma}(p) = \inf_{x \in \Sigma} \|x - p\| \tag{3-1}$$

Em nosso caso particular, estamos interessados em calcular um campo de distância para determinar a menor distância de cada *voxel* do nosso volume para a isosuperfície ∂V. É importante para nós que essas distâncias calculadas contenham um sinal, indicando se elas são interiores ou exteriores a ∂V. Logo, a função de distância que realmente nos interessa é

$$dist_{V}(p) = sinal(p) \inf_{x \in \partial V} \|x - p\| \tag{3-2}$$

onde

$$sinal(p) = \begin{cases} -1, & \text{se p interior à } \partial V \\ +1, & \text{caso contrário} \end{cases} \tag{3-3}$$

A maneira trivial de calcular essas distâncias é implementando um algoritmo de força-bruta, onde todas as distâncias entre *voxels* são computadas para garantidamente encontrar a menor distância para cada um dos *voxels*. Embora seja uma solução simples, é um algoritmo que pode ser muito caro computacionalmente dependendo do tamanho do volume de entrada (sua complexidade é $O(n^2)$).

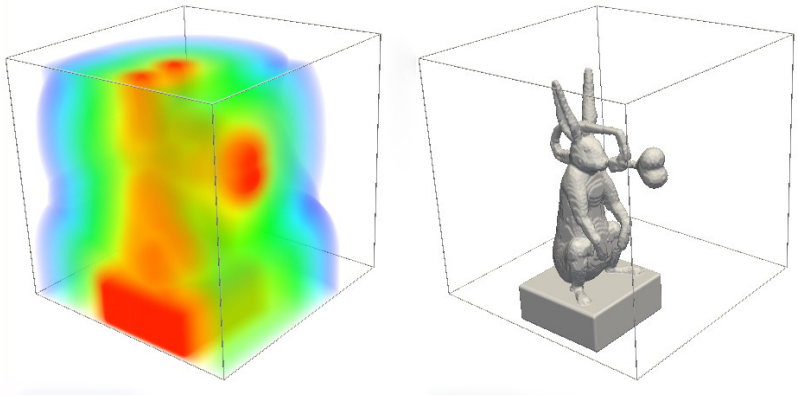

Figura 3.2: Resultado do cálculo de campo de distância utilizando uma transformada de distância em um volume segmentado à esquerda e sua respectiva isosuperfície criada à direita.

Com intuito de poder calcular campos de distância de maneira mais rápida, foram propostos outros métodos, como as transformadas de distância. O princípio por trás de uma transformada de distância é a propagação

de valores de distância pré-computados para alguns *voxels* de um volume (normalmente os *voxels* que definem e separam o que está dentro de ∂V do que está fora) para o restante dos *voxels*. Embora sejam computacionalmente mais leves, esses métodos por não calcularem uma distância exata entre *voxels* podem introduzir erros numéricos. A Figura 3.2 ilustra o resultado de uma transformada de distância em torno de um objeto segmentado em um dado binário.

3.4 Octree

Octrees [Meagher, 1982] são árvores que representam espacialmente e hierarquicamente um volume tridimensional, onde cada nó da árvore representa uma subregião do volume. Uma *octree* pode ser criada a partir de um nó raiz e, dado algum critério de subdivisão de nós, é dividido dando origem sempre a oito novos filhos, como exemplifica a Figura 3.3. Uma *octree* também pode ser criada a partir de uma subdivisão regular de um volume, onde as células folhas são colapsadas segundo alguma métrica, sempre em conjunto de oito células irmãs hierarquicamente. Em nossa implementação, utilizamos uma construção *top-bottom*, isto é, a partir da subdivisão inicial de um nó raíz.

Octrees são de extrema importância na criação de soluções de rendering ou extração de superfícies em multi-resolução, pois permitem refinar a visualização apenas em regiões de maior interesse, como por exemplo, partes do volume que estão mais próximos da câmera em caso de uma visualização volumétrica, ou também maior subdivisão de áreas de uma superfície segmentada

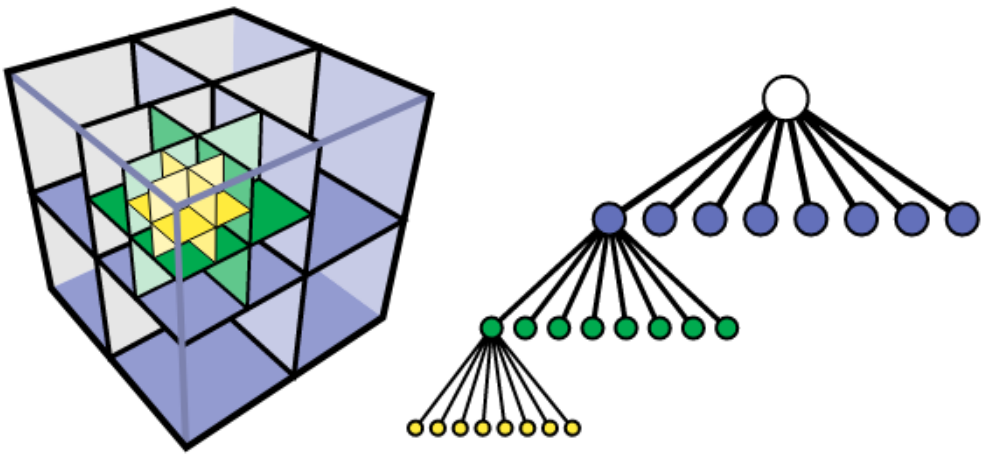

Figura 3.3: Exemplo de decomposição de um volume em uma *octree* de forma *top-bottom*. Fonte: [Engel et al., 2004]

dentro de um volume que possuem um grau de curvatura mais alto que o restante.

3.4.1 Células

Uma *octree* é representada por um conjunto de células. As células são cubos que representam espacialmente uma determinada região do volume. Uma célula também pode ser vista como um subconjunto dos *voxels* contidos no volume na região a qual ela representa. Numa representação binária de um volume previamente segmentado, temos apenas dois valores. Os *voxels* internos a ∂V são identificados por 0 e *voxels* externos são identificados por 1.

Células que possuem todos seus os seus seis voxels de *corner* com apenas um desses valores são chamadas de células homogêneas, ou células não-heterogêneas. Células que possuem pelo menos um *corner* de valor diferente dos restantes são chamadas de células heterogêneas. Se uma célula é heterogênea, ela possui arestas bipolares, ou seja, arestas que possuem *voxels* em seus extremos com valores diferentes. Arestas bipolares indicam que a isosuperfície intercepta a célula na aresta em questão pelo menos uma vez. A Figura 3.4 ilustra uma configuração de células adjacentes que possuem arestas bipolares. É importante definir a nomenclatura para esses tipos diferentes de células, pois precisamos identificar futuramente no algoritmo esses tipos de células.

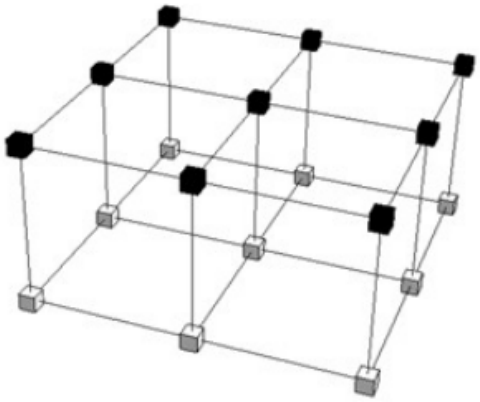

Figura 3.4: Ilustração de quatro células heterogêneas adjacentes onde todos os *voxels* nos *corners* são identificados por valores binários. Todas as arestas verticais são bipolares, uma vez que os *voxels* contidos em seus extremos possuem valores distintos.

3.4.2 Codificação de Morton

Para identificar nossas células espacialmente e hierarquicamente dentro de nossa estrutura de *octree*, utilizamos a codificação de Morton [Morton, 1966]. O código Morton de uma chave k_n da célula n pode ser construído recursivamente a partir da hierarquia dentro da *octree*, como também pode ser computado a partir da localização espacial da célula n.

Ao longo da construção da árvore, nós calculamos e salvamos as células e seus códigos de Morton utilizando o primeiro método. A chave do nó raíz da *octree* é 1, as chaves dos nós filhos são uma concatenação de 1 com o índice do octante que o filho representa escrito de forma binária. Um nó filho sempre terá como chave a chave de seu nó pai concatenada com a representação binária do octante a que pertence, ou seja, cada nível da árvore incrementa três *bits* em relação ao nível anterior (Figura 3.5).

Por exemplo, uma célula de chave 97 e uma célula filha que pertence ao octante três. A representação binária desta chave é 1 100 001. Para obter a chave da célula filha, concatenamos esses *bits* com os *bits* do octante, ou seja, teremos como chave 1 100 001 **011**, por exemplo. Consequentemente, se quisermos encontrar a chave da célula pai k_{pai} de uma célula em particular k, basta fazer uma operação binária de *shift*, ou seja, $k_{pai} = k \gg 3$.

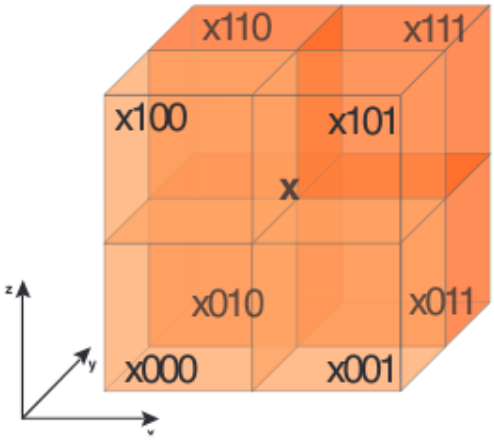

Figura 3.5: Sufixos binários para construção da chave de células filhas da célula de chave x. Fonte: [Lewiner et al., 2010]

4
Método desenvolvido

Neste capítulo é apresentado uma descrição detalhada do *pipeline* do método desenvolvido. É esperado de entrada um dado volumétrico binarizado previamente segmentado por outra ferramenta. O nosso método, uma reprodução modificada do método [Lobello et al., 2014], pode ser abstraído como três etapas. Essas etapas são resumidas a seguir e detalhadas nas seções seguintes.

Pré-processamento Nesta etapa, o dado binário é processado por um método de cálculo de campo de distância. Esse campo de distância, então, é dividido em subvolumes que são persistidos separadamente em disco rígido (ou em uma memória secundária qualquer). Estes subvolumes são chamados de meta-células.

Processamento de meta-células Esta etapa é responsável por processar individualmente cada meta-célula, onde cada um desses processamentos é responsável por criar uma parte da superfície de saída.

Conexão de meta-células É a última etapa do *pipeline*. Essa etapa é necessária para conectar as superfícies da etapa anterior, criando toda a geometria que as conectam.

4.1
Pré-processamento

Um dado volumétrico binário ainda não é o tipo de entrada ideal para calcular uma isosuperfície de qualidade. Como os valores escalares do volume são usados para o cálculo geométrico e topológico da isosuperfície de saída, e esses cálculos utilizam interpolações lineares, possuir apenas valores binários para determinar se um determinado voxel está dentro ou fora da superfície segmentada faz com que não seja criada uma superfície suave. Ao invés disso, têm-se uma superfície com diversos desníveis. Para contornar esse problema e obter uma superfície mais suave, calcula-se um campo de distância a partir do volume binário.

Além disto, existe também o problema da disponibilidade de memória RAM da máquina. Para contornar esse segundo problema, é adotada uma estratégia *out-of-core* com intuito de utilizar também o disco rígido para disponibilizar os dados necessários para a extração da isosuperfície.

Para calcular nosso campo escalar, implementamos o método de [Felzenswalb et at., 2012]. A partir deste campo de distância, utilizamos uma estratégia inspirada na descrita em [Lobello et al., 2014] em nosso método para persistir os dados em disco. A estratégia se baseia em dividir o volume em sub-volumes chamados meta-células. Estas meta-células são, então, persistidas em disco em arquivos separados (Figura 4.1).

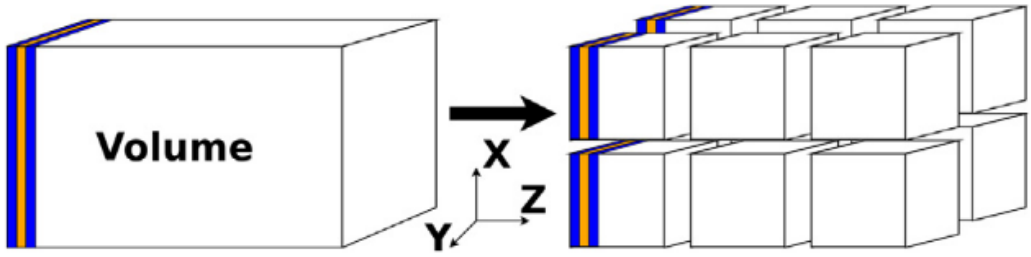

Figura 4.1: Exemplo de subdivisão de um volume em 12 meta-células. Fonte: [Lobello et al., 2014]

Como é conhecida a estrutura interna do dado binário escrito em disco, é possível ler diretamente do disco quaisquer *voxels* que se deseja saber o valor. Dessa forma, uma meta-célula de cada vez é lida do disco e levada à memória para que possa ser organizada espacialmente dentro de um novo *buffer*, que por sua vez é escrito em um arquivo temporário. Todas meta-células dividem entre si os mesmos *voxels* em faces que são adjacentes entre si.

Embora nosso método atualmente faça este processo de forma serial, ou seja, extraindo uma meta-célula de cada vez, esse processo pode ser paralelizado. O volume escalar calculado é usado apenas para leitura, logo com algum tratamento de *data racing* é possível fazer essa separação de forma mais rápida, se a máquina possuir memória suficiente para isto.

4.2 Processamento de meta-células

Uma vez que todas as meta-células estão persistidas em disco, é necessário processá-las para extrair as superfícies parciais. Para calcular a superfície de uma meta-célula, criamos primeiro uma *octree* a partir deste sub-volume. Criamos a *octree* de forma *top-bottom* utilizando um critério topológico que

subdivide a árvore a partir de um nível. Após a criação da *octree*, processamos as suas células folhas, identificando as que são interceptadas por ∂V e criando vértices representativos para essas células, chamados vértices duais. Conectamos esses vértices duais adjacentes, seguindo um critério. Ao final desse procedimento, cada meta-célula possui sua respectiva parte da malha criada (Figura 4.2). Cada um desses passos está detalhado nas subseções seguintes.

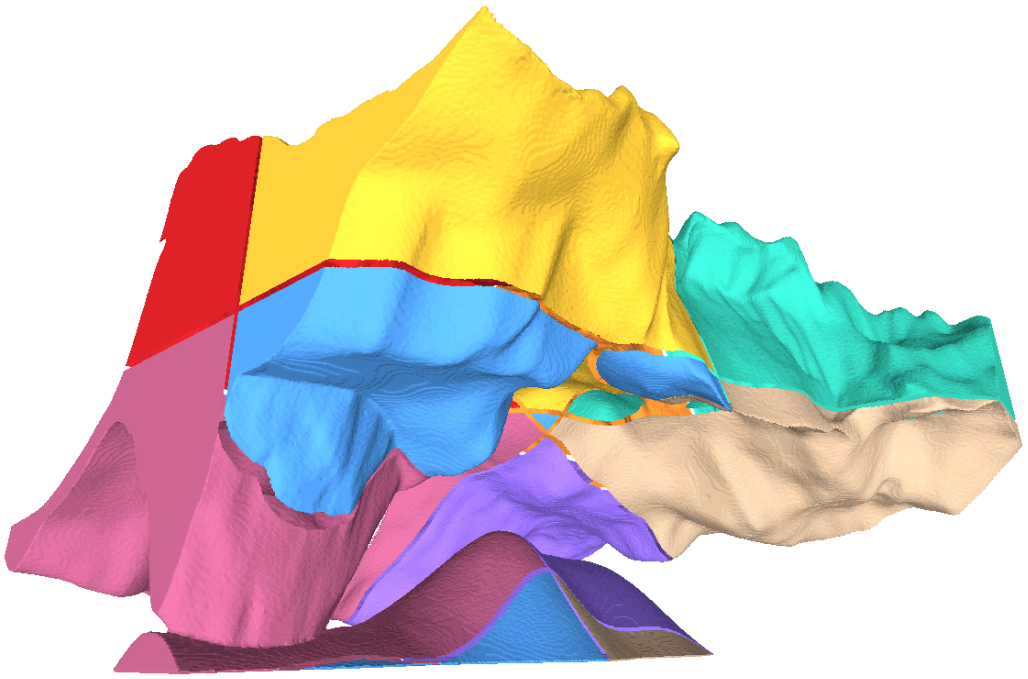

Figura 4.2: Isosuperfície extraída pelo processamento de 8 meta-células. Cada meta-célula processada apresenta uma coloração diferente. As áreas de conexão de meta-células ainda não foram processadas neste ponto.

4.2.1 Construção de *octree*

Como em [Lobello et al., 2014], a *octree* é construída e persistida de forma linear. Cada célula é identificada por uma chave única, utilizando a codificação de Morton [Morton, 1966]. Esta codificação das células em memória permite identificar qualquer célula hierarquicamente e espacialmente, fornecendo fácil e rápido acesso às células vizinhas ou hierarquicamente conectadas.

A *octree* é criada a partir de uma subdivisão regular inicial do dado respeitando um parâmetro de nível mínimo da árvore. Entre o nível máximo, também parametrizado, e o nível mínimo, as células são subdivididas seguindo um critério topológico.

Este critério topológico serve para, além de subdividir a árvore adaptativamente respeitando a topologia contida na superfície segmentada, criar o que os autores chamam de *octree* topologicamente segura. Quando se tem uma *octree* topologicamente segura, é garantido que não existem arestas ou vértices não *manifold* na superfície extraída. O critério topológico é baseado na definição de célula complexa (Figura 4.3).

- Seja C uma célula folha da *octree*, C é complexa se pelo menos uma de suas arestas ou faces é complexa.
- Uma aresta é complexa se ela é interceptada por ∂V mais de uma vez.
- Uma face é complexa se intecepta ∂V sem conter nenhuma aresta complexa mas possui uma componente conexa isolada a atravessando.

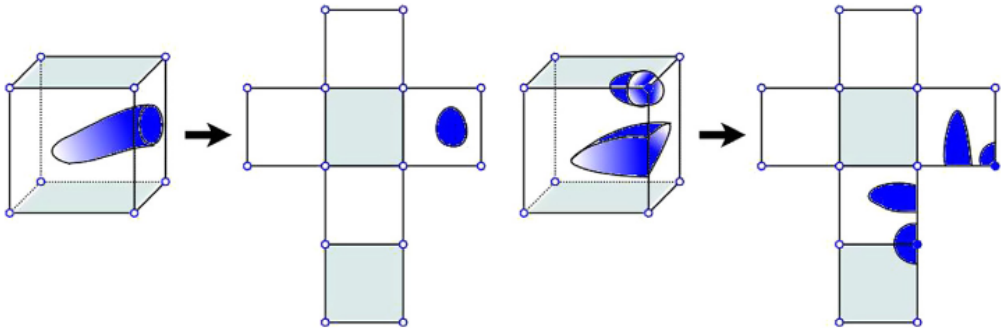

Figura 4.3: Exemplos de células complexas, onde uma superfície ∂V atravessa. Célula da esquerda é um exemplo de face complexa, enquanto a da direita exemplifica uma célula uma aresta complexa. Fonte: [Lobello et al., 2014]

Para identificar uma aresta complexa, percorremos sequencialmente os *voxels* que pertencem à aresta e identificamos se ocorre mudanças de valor mais de uma vez. Para identificar componentes conexas em faces complexas, aplicamos um algoritmo conhecido como Hoshen-Kopelman [Hoshen and Kopelman, 1976] sobre os *voxels* contidos nestas faces. Exemplificamos casos de células complexas na Figura 4.3.

Nosso algoritmo para identificar se existe uma componente conexa isolada em uma face se baseia em percorrer os *voxels*, identificando os que são internos à ∂V. Se um *voxel* é interno, não possui um *label* associado ainda e também não possui adjacência com nenhum outro *voxel* que possua *label* associado, um novo *label* é criado para ele. Caso o *voxel* possua um *voxel* vizinho com um *label* já definido, significa que eles pertencem à mesma componente conexa, logo o mesmo *label* é associado a este elemento.

Após identificar todas as componentes conexas, se existir uma componente onde nenhum de seus *voxels* pertence à uma aresta da face em questão, então temos uma face complexa.

Seja uma célula C complexa de acordo com a definição anterior, ela deve ser subdivida, segundo [Lobello et al., 2014]. Em [Lobello et al., 2012], porém, existe uma regra adicional para subdivisão de uma célula que não é citada em [Lobello et al., 2014]. Caso uma célula seja não-homogênea, ou seja, células que possuem pelo menos um vértice de valor diferente dos demais, ela deve ser subdividida também.

Quando a árvore é criada seguindo apenas a regra de subdivisão de células complexas, possuímos células interceptadas por ∂V em diversos níveis de refinamento. Optamos, porém, por criar uma árvore balanceada para ter uma superfície de melhor qualidade. Em nossa árvore, neste caso, as células folhas terão no máximo um nível de profundidade de diferença entre si.

Ao combinar a regra de células complexas com a regra de células não-homogêneas, é criada uma *octree* onde todas as células interceptadas por ∂V estão no mesmo nível, no caso, o maior nível de refinamento parametrizado.

Cada uma dessas árvores criadas tem suas particularidades e vantagens de serem utilizadas. Como cada árvore resultante é sensível ao critério adotado, o novo ou o antigo, apresentamos separadamente as características de cada uma das construções a seguir. Por conveniência, vamos nos referir à construção segundo [Lobello et al., 2012] como **construção regular** e à construção segundo [Lobello et al., 2014] como **construção adaptativa**.

4.2.2 Localização de vértices duais

Antes de descrever como os vértices duais vão ser conectados entre si, vamos primeiro explicar como localizá-los. Assim como em [Ju et al., 2002], o cálculo da posição dos vértices duais é a mesma para os dois tipos de construção de *octree*. Apenas células interceptadas por ∂V possuem um vértice dual associado, essas células então tem pelo menos uma de suas arestas bipolares.

O cálculo é realizado através de um cálculo de minimização de uma função de erro quadrático [Garland and Heckbert, 1998]. A função de erro é montada utilizando as pontos de interseção das arestas interceptadas e as normais nessas posições:

$$E[x] = \sum_i (n_i \cdot (x - p_i))^2 \tag{4-1}$$

Nesta equação, i é um índice de cada ponto de interseção de uma aresta interceptada por ∂V, e p_i e n_i são, respectivamente, a coordenada e o valor de

normal neste ponto de interseção. Esta equação pode ser representada de forma matricial como solução pelo método dos mínimos quadrados para $Ax = b$, onde A é uma matriz $n \times 3$ onde suas linhas são preenchidas por n_i e b é um vetor composto pelos valores de $n_i \cdot p_i$.

Para encontrar os valores de p_i, nós percorremos os *voxels* pertencentes às arestas interceptadas por ∂V e identificamos os dois *voxels* de transição. Calculamos uma interpolação linear entre estes dois voxels, usando suas coordenadas e seus valores escalares.

Para encontrar os valores de n_i, aplicamos o operador de Prewitt sobre esses *voxels*. Este operador é, tecnicamente, um operador de diferença e calcula uma aproximação do gradiente em um determinado ponto aplicando máscaras $3 \times 3 \times 3$ que são convolucionadas com o volume.

Assim como em [Schaefer and Warren, 2003], resolvemos $Ax = b$ achando a solução x minimizante, minimizando em direção ao centro de massa da célula. O centro de massa é a média das coordenadas de interseção nas arestas da célula. Calculamos esta coordenada computando o valor de $x = c + p$, sendo p a coordenada do centro de massa da célula e

$$c = (A^T A)^+ (A^T b - A^T A p) \tag{4-2}$$

Para calcular c, precisamos calcular a inversa de Moore-Penrose $(A^T A)^+$ da matriz $A^T A$. Esta inversa também conhecida como pseudo-inversa, primeiramente introduzida em [Moore, 1920], é computada a partir de uma decomposição em valores singulares da matriz $A^T A$. Como o resultado do problema de minimização pode ser uma coordenada que não esteja dentro dos limites espaciais da célula, nestes casos nós usamos como posição para o vértice dual o próprio centro de massa p.

4.2.3 Conectividade

Para conectar os vértices duais, optamos por implementar o algoritmo de [Lobello et al., 2014]. O algoritmo consiste em percorrer de forma sequencial cada uma das células da *octree*, verificando se ela é interceptada por ∂V e se ainda não foi processada.

Se a célula ainda não foi processada, suas arestas interceptadas são processadas. Uma vez que uma aresta é identificada e processada, ela é marcada como tal tanto na célula atual sendo processada, quanto nas células vizinhas a qual ela também pertence. Se as células vizinhas não forem todas células folhas, essa aresta não é processada. Quando todas as arestas interceptadas da célula são processadas, ela é marcada como processada.

Por consequência desse controle na identificação de arestas processadas e a quais células elas pertencem, algumas células não visitadas já podem ter sido dadas como processadas por já terem tido suas arestas processadas por outras células vizinhas.

Algorithm 4.1: Conectividade de células

```
   Input  : Lista L de células folhas da octree
   Output: Conjunto de vértices duais V
   Output: Conjunto de triângulos F
 1 V ⟵ ∅
 2 F ⟵ ∅
 3 foreach c ∈ L do
 4     if ¬celulaMarcada(c) ∧ celulaFolha(c) then
 5         arestas ⟵ arestasBipolares(c)
 6         foreach e_i ∈ arestas do
 7             if ¬arestaMarcada(e_i) then
 8                 celulas ⟵ celulasCoincidentes(e_i)
 9                 for i = 0 ⟶ 4 do
10                     if celulaFolha(celulas(i)) then
11                         vDuais(i) ⟵
                            calcularVerticeDual(celulas(i))
12                     end
13                     else
14                         continuar
15                     end
16                 end
17                 V ⟵ V ∪ vDuais
18                 F ⟵ F ∪ criarTriangulacao(vDuais)
19                 marcarAresta(e_i)
20             end
21         end
22         if todasArestasMarcadas(arestas) then
23             marcarCelula(c)
24         end
25     end
26 end
```

A construção regular e a construção adaptativa da *octree* resultam em duas árvores com particularidades distintas. Embora o mesmo algoritmo de conectividade seja utilizado para as duas construções, por serem tão diferentes, é adequado o uso de métodos diferentes para o posicionamento e conexão dos vértices duais nos dois casos. No caso regular, nós podemos posicionar mais de um vértice dual dentro de uma célula e utilizamos o *Dual Marching Cubes* para poder posicionar estes pontos. No caso adaptativo, posicionamos apenas um vértice dual por célula.

4.2.3.1
Construção regular

Com o algoritmo de conectividade, identificamos quais são as arestas interceptadas por ∂V e a quais células folhas essa aresta pertence. Nesta *octree* todas as arestas internas vão pertencer a exatamente quatro células adjacentes.

Como sugerido em [Lobello et al., 2014], utilizamos [Nielson, 2004] para conectar e posicionar os vértices duais. O *Dual Marching Cubes* é um método dual concebido para funcionar em uma grade regular e, diferente de [Ju et al., 2002], permite posicionar mais de um vértice dual dentro de uma célula. Mesmo com essa flexibilização no número de vértices possíveis de serem inseridos dentro dos limites espaciais de uma célula, ele ainda pode apresentar arestas não *manifolds* em determinados casos, mas esse problema é facilmente contornável com um tratamento simples, explicado mais à frente.

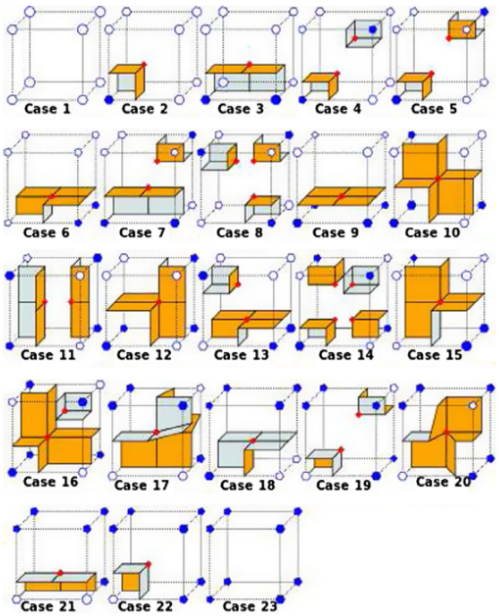

Figura 4.4: Tabela de consulta do algoritmo de *dual marching cubes* [Nielson, 2004]. A tabela completa possui 256 entradas, todas as entradas são derivações dos 23 casos bases. Fonte: [Lobello et al., 2014]

O primeiro passo do algoritmo de [Nielson, 2004] é identificar os valores nos oito vértices de uma célula. Com esses oito valores, é possível montar uma chave de oito *bits* (ou seja, os valores de chave podem variar entre zero e duzentos e cinquenta e cinco). Essa chave serve para acessar uma tabela pré-computada (Figura 4.4), que entrega como resultado quais são as arestas interceptadas desta configuração que devem ser usadas para calcular a posição do vértice dual. Isto acontece pois o *Dual Marching Cubes* pode posicionar mais de um vértice dual por célula, então cada possível possível vértice dual de uma

determinada configuração tem que estar associado a um conjunto distinto de arestas bipolares.

Para cada uma das quatro células adjacentes que contêm a aresta em questão, deve-se montar uma chave, acessar a tabela de consulta, calcular a posição do vértice dual e conectar esses quatro vértices entre si, construindo dois triângulos.

Para tratar os casos ambíguos que levam a construção de arestas não *manifolds*, deve-se identificar para toda célula se sua configuração é $2B$ ou $3B$ e se, além disso, essas células possuem sua face ambígua compartilhada com outra célula de configuração $2B$ ou $3B$. Nesses casos específicos, para essas células, deve-se retornar uma configuração alternativa, conforme ilustra a Figura 4.5.

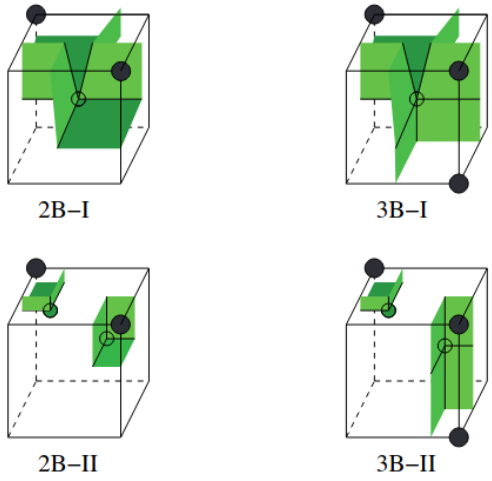

Figura 4.5: As configurações 2B-I (equivalente ao caso 4 na tabela anterior) e 3B-I (equivalente ao caso 7), ao dividirem uma face ambígua, devem utilizar uma configuração alternativa de arestas para que não sejam criadas arestas ou vértices não manifold. Fonte: [Wenger, 2013]

4.2.3.2 Construção adaptativa

A construção adaptativa constrói uma *octree* onde existem células interceptadas em diferentes níveis de resolução. Neste caso, não podemos utilizar o método anterior para posicionar os vértices duais, uma vez que [Nielson, 2004] foi concebido para funcionar apenas em grades regulares.

Para posicionar os vértices duais e conectá-los neste caso, utilizamos o método em [Ju et al., 2002]. Neste método, apenas arestas mínimas (Figura 4.6), onde a superfície ∂V cruza, são processadas e criam polígonos. Arestas mínimas são arestas que não contêm propriamente outras arestas menores que a mesma.

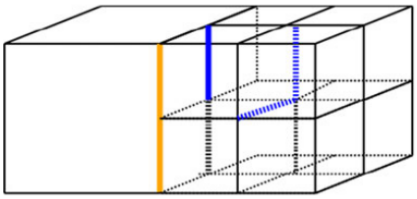

Figura 4.6: Exemplos de arestas mínimas. Arestas mínimas (em azul) não contém outras arestas. Em amarelo temos um exemplo de uma aresta que não é mínima. Fonte: [Lobello et al., 2014]

Quando uma aresta mínima está sendo processada, são localizadas as células que as contêm, conectando os vértices duais dessas células entre si, formando uma triangulação trivial. Em um caso regular, a aresta é contida em quatro células, podendo essas células ter níveis diferentes de refinamento, criando dois triângulos. Em um caso adaptativo, a aresta é contida por três células diferentes, onde garantidamente uma das células vai estar em um nível de refinamento menor, criando apenas um triângulo. A Figura 4.8 uma triangulação no caso regular, a Figura 4.9 exemplifica uma triangulação no caso adaptativo.

Essa construção de *octree*, assim como na construção regular, não garante que a superfície construída é livre de interseção de polígonos. Porém, utilizando uma extensão de [Ju et al., 2002], podemos tratar esses casos.

Segundo [Ju et al., 2006], a maneira mais simples de obter uma isosuperfície livre de interseções em uma grade adaptativa de uma *octree* é um aplicar um método híbrido entre um método dual e um método primal. Um método primal cria os vértices de sua superfície os posicionando nas arestas interceptadas, enquanto método duais posicionam seus vértices dentro dos limites das células.

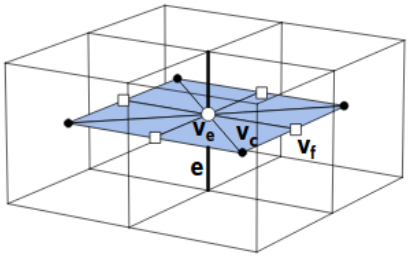

Figura 4.7: Um leque de triângulos criado pelo método híbrido a partir de uma aresta e. O vértice v_e é conectado aos vértices v_f contidos nas faces adjacentes das células que contém e e aos vértices duais v_c, localizados dentro dos limites das células. Fonte: [Ju et al., 2006]

Neste método híbrido, essas duas abordagens são combinadas. Para criar os polígonos, ao invés de criar uma triangulação trivial usando os vértices duais referentes à aresta sendo processada, devemos criar um leque de triângulos a partir de e onde cada triângulo conecta v_e com os vértices v_f em faces que contêm e e os vértices duais das células v_c. Com este método, leques de oito ou seis triângulos serão criados para cada aresta mínima (ilustrado na Figura 4.7).

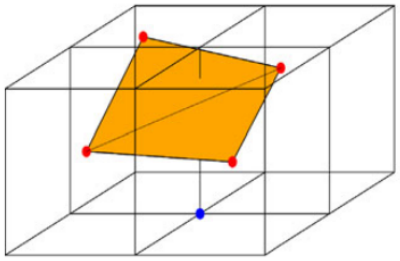

Figura 4.8: Triangulação de uma aresta mínima no caso regular. A aresta é contida por quatro células vizinhas, que foram entre si dois triângulos ligando seus vértices duais. Fonte: [Lobello et al., 2014]

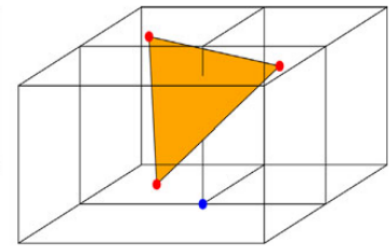

Figura 4.9: Triangulação de uma aresta mínima no caso adaptativo. A aresta é contida por três células, onde uma das células está em um nível de resolução diferente. Neste caso, apenas um triângulo é formado conectando os vértices duais. Fonte: [Lobello et al., 2014]

Embora seja um método que funcione, ainda não é o ideal pela quantidade excessiva de polígonos que acabam sendo criados para cada leque. Com intuito de diminuir a quantidade de polígonos criados por este método, [Ju et al., 2006] apresentam regras seguindo algumas verificações geométricas que permitem que uma triangulação trivial possa ser muito mais utilizada. A ideia é substituir leques de triângulos por um número menor de triângulos que ainda estão contidos dentro de um envelope válido.

4.3 Conexão de meta-células

Ao processar as meta-células individualmente, as arestas de borda não são processadas. Estas arestas não são processadas pois pertencem à células de meta-células diferentes, e meta-células não possuem acesso compartilhado de dados de meta-células vizinhas. Além das arestas de borda, também não processamos toda e qualquer aresta que seja compartilhada por uma célula de borda. Nós fazemos isso porque não podemos calcular os valores exatos dos vértices duais em células de borda na etapa anterior. Isto acontece pois, se ela possui alguma de suas arestas de borda interceptada, seria necessário aplicar o operador de Prewitt nos *voxels* e possivelmente seria necessário valores de *voxels* que não pertencem à meta-célula que a célula está.

Para processar estas arestas nesta última etapa, apenas precisamos guardar informação das células que possuem pelo menos uma face contida em alguma face externa de sua respectiva meta-célula e também todas as células que são adjacentes a essas células. Ao fim de extração de cada meta-célula, descartamos todas células que não vão ser mais necessárias nesta etapa.

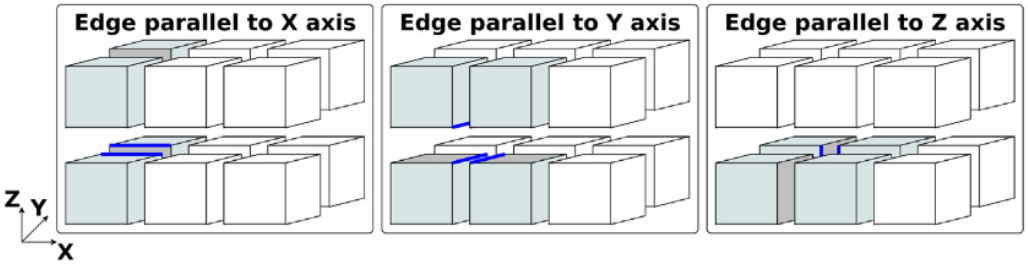

Figura 4.10: Todas as possíveis combinações de meta-células para a construção de uma *edge-octree*. Fonte: [Lobello et al., 2014]

Para conectar essas componentes de malha poligonal criadas para as meta-células, precisamos processar as arestas restantes que ainda não foram processadas. Para isso, criamos uma estrutura de dados nova, chamada *edge-octree*. Uma *edge-octree* é uma *octree* parcial que possui apenas células adjacentes contidas em faces de meta-células adjacentes entre si através de uma meta-aresta.

Primeiro, para cada meta-célula, vamos identificar as arestas que pertencem a outras três células. Cada uma dessas arestas pertence à faces que são compartilhadas entre essas quatro meta-células. Essa relação de adjacência pode ser consultada através da Tabela 4.1 e da Figura 4.10.

Eixo de Referência	Meta-célula	Código morton
Aresta paralela ao eixo X	$B_{i,j,k}$	000
	$B_{i,j+1,k}$	0**1**0
	$B_{i,j,k+1}$	**1**00
	$B_{i,j+1,k+1}$	**11**0
Aresta paralela ao eixo Y	$B_{i+1,j,k}$	000
	$B_{i+1,j,k}$	00**1**
	$B_{i,j,k+1}$	**1**00
	$B_{i+1,j,k+1}$	**1**0**1**
Aresta paralela ao eixo Z	$B_{i+1,j,k}$	000
	$B_{i+1,j,k}$	00**1**
	$B_{i,j+1,k}$	0**1**0
	$B_{i+1,j+1,k}$	0**11**

Tabela 4.1: Adjacência entre meta-células a partir de uma meta-célula de referência identificada localmente por (i,j,k). A adjacência também atribui um nível hierárquico extra para células que são inserida na *edge-octree*. Tabela reescrita de [Lobello et al., 2014].

Exemplificando, temos uma meta-aresta E_m paralela ao eixo Y e que pertence à quatro meta-células distintas $B_{i,j,k}$, $B_{i+1,j,k}$, $B_{i,j,k+1}$ e $B_{i+1,j,k+1}$. Para criar uma *edge-octree* a partir desta configuração, extraímos o conjunto de células que pertencem às meta-faces que contêm E_m.

Essas células extraídas, embora sejam adjacentes espacialmente, elas hierarquicamente ainda não estão conectadas. Para processar a *edge-octree*, nós criamos um nível extra de profundidade para essas células de acordo com a tabela 4.1. Este processo de construção da *edge-octree* é ilustrado na Figura 4.11 para uma compreensão mais fácil.

Seja uma célula C_1 indexada pelo código de Morton 1 101 e que pertence à meta-célula de indexação local (i, j, k), seu código Morton será alterado para 1 **000** 101. Seja C_2 uma outra célula adjacente a C_1 com código de Morton 1 100, ou seja e pertence à meta-célula indexada localmente por $(i + 1, j, k)$, seu código de Morton será alterado para 1 **001** 100.

Para cada *edge-octree* construída, nós então utilizamos o mesmo algoritmo de extração de superfície da seção anterior (Algoritmo 4.1) para criar os polígonos restantes e fechar a malha poligonal.

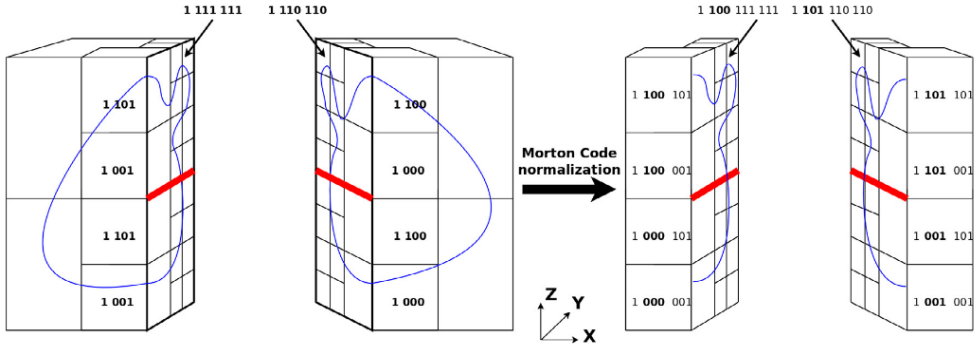

Figura 4.11: Ilustração da construção de uma *edge-octree* a partir da aresta E_n em vermelho. A superfície ∂V é o contorno em azul. Apenas as células contidas nas faces que contém E_n são selecionadas para construção da *edge-octree*. Os códigos de Morton dessas células são alterados e ganham um nível extra de profundidade, o que faz com que suas chaves sejam alteradas a partir da concatenação de suas chaves originais com três *bits* das meta-células que as contém. Fonte: [Lobello et al., 2014]

5
Resultados

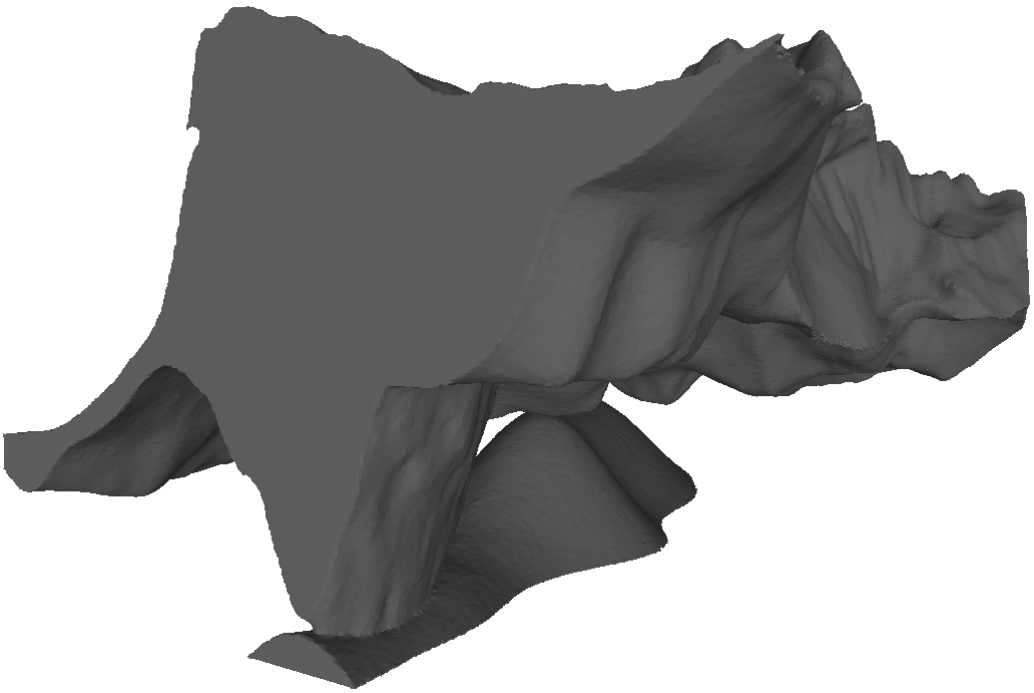

Figura 5.1: Isosuperfície do domo de sal extraída com nosso método utilizando uma construção adaptativa de nível máximo 8.

Apresentamos neste capítulo alguns dos nossos resultados com o método que implementamos. O nosso método foi desenvolvido utilizando a linguagem de programação C++. A máquina utilizada nos testes possui processador AMD Ryzen 5 1600 3.2GHz e 16GB de memória RAM disponíveis. Utilizamos também a biblioteca Eigen [Guennebaud et al., 2010] para realizar os cálculos algébricos do posicionamento dos vértices duais.

Utilizamos ao longo do desenvolvimento diversos dados sintéticos de superfícies implícitas, porém a maior parte dos testes foram utilizando o dado volumétrico binário de domo de sal. Esse dado, que ocupa cerca de 2GB de espaço em disco, é resultado de um desafio de segmentação realizado pela TGS, e é uma fatia do dado original que possui uma resolução muito maior. Uma superfície extraída deste dado usando nossa implementação pode ser vista na Figura 5.1. O dado binário utilizado possui dimensões $891 \times 951 \times 601$,

porém inserimos por conveniência dentro de um volume de dimensões $1025 \times 1025 \times 1025$ preenchido apenas com voxels de valor igual a zero. Todos nossos testes foram utilizando uma subdivisão do volume binário em 8 meta-células de dimensões $513 \times 513 \times 513$.

5.1 Pré-processamento

Nossa etapa de pré-processamento pode ser divida em sub-etapas:

- Leitura do volume
- Inicialização do campo de distância
- Cálculo do campo de distância
- Cálculo do sinal das distâncias
- Divisão das meta-células

Em nosso método, experimentamos duas formas de calcular o campo de distância. O primeiro método que implementamos foi o [Satherley and Jones, 2001]. Este método é uma transformada de distância de oito passadas, não é paralelizável e além disso calcula uma aproximação do campo de distância que pode não dar resultados muito bons por conta de erros propagados em algumas configurações de *voxels*.

O segundo método, o método que adotamos para nossa solução, é o de [Felzenswalb et at., 2012]. Este método calcula em apenas 3 passadas um campo de distância com todas as menores distâncias euclidianas entre os *voxels* do volume, sem introdução de erros de propagação ou numéricos (Figura 5.2). Além disso, cada uma dessas passadas são paralelizáveis, o que torna o método muito mais eficiente em relação ao tempo de processamento comparado com [Satherley and Jones, 2001].

Das sub-etapas mencionadas antes, apenas a cálculo do campo de distância é afetado pela escolha dos métodos mencionados, em termos de tempo de processamento. Ao utilizar [Satherley and Jones, 2001], levamos 722 segundos para obter uma aproximação do campo escalar de distâncias. Se utilizarmos [Felzenswalb et at., 2012], esse tempo de processamento diminui para 270 segundos, uma redução de 63%. Ao paralelizarmos este cálculo, temos uma redução de tempo de processamento ainda mais significativa, o tempo diminui de 270 para 47 segundos. No final das contas, temos uma redução de aproximadamente 93% no tempo de processamento entre os dois métodos. Esta melhoria no tempo de processamento, porém, vem em troca de um consumo maior de memória.

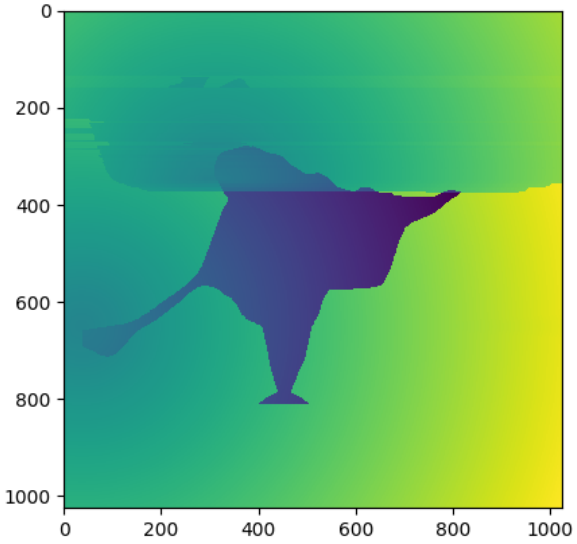

Figura 5.2: Fatia 2D de dimensões 1025×1025 do campo de distância calculado a partir de nossa implementação de [Felzenswalb et at., 2012]. Como pode ser visto, as bordas são muito bem definidas por se tratar de um método exato de cálculo de distâncias euclidianas e não incluir erros de propagação de valores, comuns em transformadas de distância.

Por fim, gastamos na leitura do volume menos de um segundo, o que é insignficativo comparado às sub-etapas seguintes. Para inicializar o campo de distância, gastamos 31 segundos. Para definir o sinal dos valores de distância, gastamos 3.7 segundos. Por fim, para a divisão do campo em meta-células e suas persistências em disco, levamos 148 segundos, nosso maior gargalo nesta etapa de pré-processamento. No total, esta etapa do nosso método leva 231 segundos para ser realizada.

5.2 Eliminação de auto-interseção em construções adaptativas

Como dito anteriormente, implementamos um tratamento geométrico ao conectar vértices duais em uma construção de *octree* adaptativa com intuito de eliminar auto-interseções da superfície construída (Figura 5.3). Discutimos aqui o impacto deste tratamento geométrico de [Ju et al., 2006] no tempo de processamento e também na quantidade de dados criados.

Fizemos alguns testes, criando duas árvores com níveis máximos diferentes, uma com nível máximo igual a 6 e outra com nível máximo igual a 7. Em ambos os casos, experimentamos criar a superfície sem o uso do tratamento geométrico e com o tratamento. A Figura 5.3 ilustra uma região da superfície do domo de sal antes e depois de aplicado o tratamento.

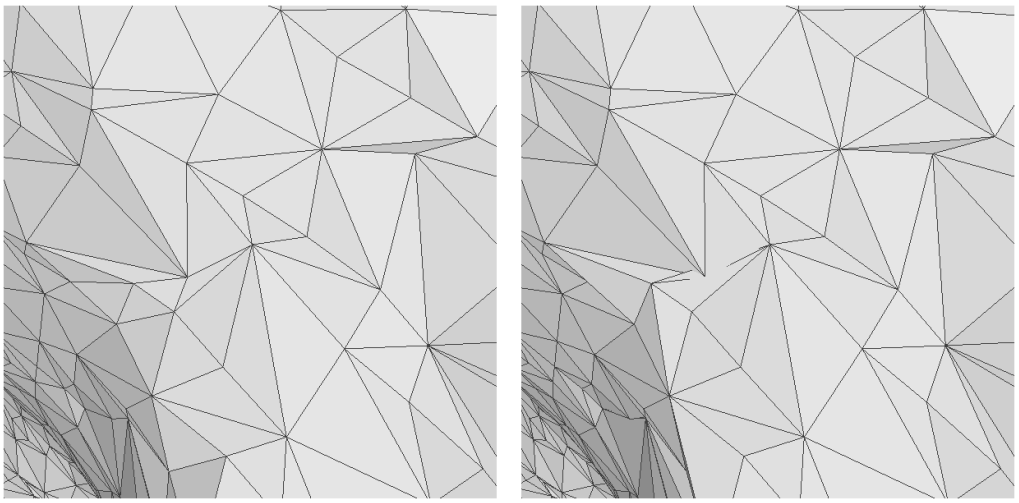

Figura 5.3: Uma região da isosuperfície do domo de sal que apresenta auto-interseção sem o tratamento de face ambígua (direita) e a mesma região sob o tratamento (esquerda).

Em termos de tempo, ao eliminarmos auto-interseções temos um pequeno aumento no tempo de processamento das meta-células. Em uma árvore de nível máximo 6, o tempo total de processamento das meta-células passou de 144 segundos para 154 segundos. Em uma árvore de nível máximo 7, o tempo total de processamento das meta-células passou de 161 segundos para 171 segundos. Na etapa de conexão de meta-células, houve um incremento de apenas 2 segundos.

Em muitas etapas das do método que implementamos, como a criação da *octree*, não houve mudança no tempo de execução pois esse tratamento geométrico apenas afeta o momento de conexão e criação dos vértices que vão compor a malha.

Ao aplicar as verificações geométricas de [Ju et al., 2006], em muitas configurações de células são criados leques de triângulos ao invés de uma triangulação trivial, o que resulta em um aumento de vértices e triângulos na superfície criada.

Em nossa árvore de nível máximo 6, sem aplicar o método de eliminação de auto-interseção, obtemos uma superfície que contém 85.220 vértices e 41.690 triângulos. Ao aplicar o método, este número sobe para 87.138 vértices e 44.596 triângulos. Já na árvore de nível máximo 7, o número de vértices criados aumenta de 299.016 para 302.839, enquanto o número de faces aumenta de 148.188 para 154.248 (Tabela 5.1 e Tabela 5.2).

Em [Ju et al., 2006], os autores discutem este aumento no número de polígonos criados e dizem que este incremento costuma ser menor do que 50%

nos casos mais complexos, ou seja, nos casos onde existe um número grande de interseções. Na árvore de nível máximo 6, temos um incremento de 6% e na árvore de nível máximo 7, encontramos um incremento de 4% apenas.

Este aumento está relacionado ao número de arestas mínimas que não foram aceitas como candidatas à uma triangulação trivial. Por exemplo, em uma única meta-célula foram processadas 1468 arestas em uma árvore de altura nível máximo 6. Dessas, apenas 96 arestas não criaram uma triangulação trivial, ou seja, criaram leques de tamanhos diferentes.

Esta grande diferença entre o número de arestas que precisaram ou não criar leques indica que a aplicação deste método é de extrema relevância para nosso resultado, pois não compromete o tempo de processamento do algoritmo e gera uma superfície correta.

Número de vértices	Sem auto-interseção	Com auto-interseção
Nível máximo 6	87.138	85.220
Nível máximo 7	302.836	299.016

Tabela 5.1: Comparação entre o número de vértices criados utilizando uma construção adaptativa aplicando o método de eliminação de auto-interseção e sem a aplicação do método.

Número de polígonos	Sem auto-interseção	Com auto-interseção
Nível máximo 6	44.596	41.690
Nível máximo 7	154.248	148.188

Tabela 5.2: Comparação entre o número de polígonos criados utilizando uma construção adaptativa aplicando o método de eliminação de auto-interseção e sem a aplicação do método.

5.3 Tratamento de ambiguidade para construções regulares

Como dito no capítulo anterior, com intuito de construir uma superfície *manifold*, precisamos identificar faces de células adjacentes que compartilham uma face ambígua e que possuem configurações 2B ou 3B da *look-up table* do algoritmo de *Dual Marching Cubes*. Ao testar em diferentes níveis máximos de árvore, percebemos que esse tratamento de ambiguidade não resultou em um aumento expressivo no tempo de processamento e conexão das meta-células. Por exemplo, utilizando uma árvore de nível máximo 6, o tratamento de ambiguidade resultou num aumento do tempo de processamento de apenas 3 segundos.

Além disso, diferente do método de eliminação de interseções em construções adaptativas, este tratamento nos permite obter uma superfície *manifold*

e ao mesmo tempo não provoca um aumento no número de vértices e faces criadas.

5.4 Construção regular x Construção adaptativa

Nesta seção, comparamos os resultados que obtivemos utilizando os modos de construção de *octree*. Primeiramente, comparamos a construção regular com a construção adaptativa em relação ao tipo de superfície que os dois constroem. Em ambos os casos, temos uma superfície fechada (ou *watertight*), sem *cracks* ou buracos. Idealmente, gostaríamos de obter uma superfície que também seja sem auto-interseção e *manifold*, porém cada uma das construções pode apenas entregar, garantidamente, um desses atributos.

Ao escolher qual desses atributos é mais importante para superfície de saída, também temos que levar em consideração o tempo de processamento que cada uma dessas construções implica. Em ambos os casos, possuímos uma implementação serial e uma implementação paralelizada do nosso método de extração de isosuperfície. Em nossa implementação serial, as meta-células são processadas em fila, enquanto na implementação paralelizada as meta-células podem ser processadas ao mesmo tempo. Implementamos estas duas formas com intuito de permitir que máquinas que tem uma maior disponibilidade de memória sejam capazes de fazer esse processamento de forma mais rápida. Testamos árvores construídas de níveis máximos 6 e 7.

5.4.1 Consumo de recursos

Ao criar uma *octree* a partir de uma **construção regular** não paralelizada, temos um tempo total de processamento de 129 segundos para a árvore de nível máximo 6 e tempo de processamento de 226 segundos para a árvore de nível máximo 7. Ao paralelizarmos o algoritmo com uso de oito *threads* (ou seja, uma *thread* exclusiva para o cálculo de cada meta-célula), temos uma redução do tempo de processamento para aproximadamente 23 segundos em uma árvore de nível máximo 6. Para a árvore de nível máximo 7, o tempo do processamento é de 78 segundos.

Em uma **construção adaptativa** não paralelizada, em uma árvore de nível máximo 6 o tempo de processamento é de 163 e, em uma árvore de nível máximo 7, o tempo de processamento é de 183 segundos. Paralelizando o algoritmo com o mesmo número de threads que a construção anterior, a árvore de nível máximo 6 tem tempo de processamento 33 segundos e a árvore de

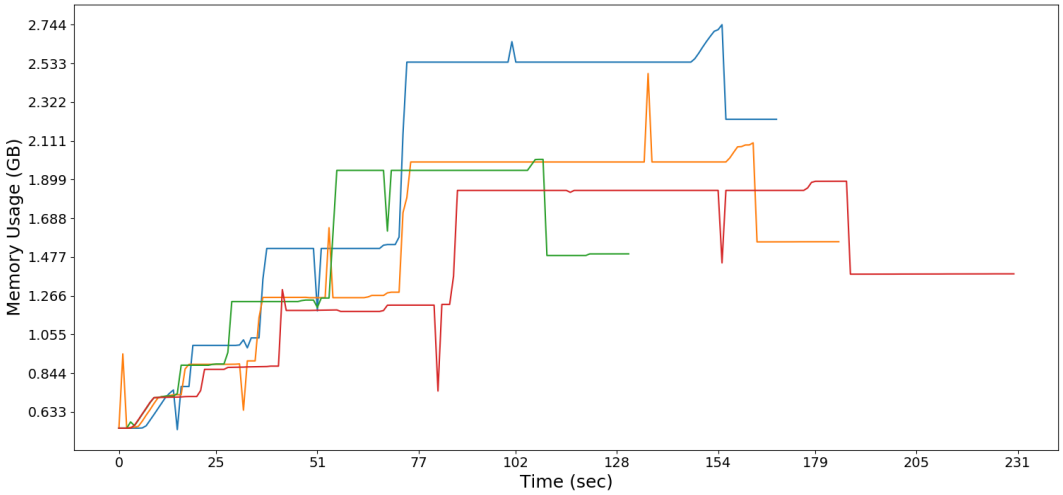

Figura 5.4: Consumo de memória dos processamentos lineares ao longo do tempo. As curvas azul e amarela são referentes à construção adaptativa (níveis máximos 6 e 7, respectivamente) e, as curvas verde e vermelho, se referem à construção regular (também respectivos aos níveis indicados anteriormente).

nível maximo 7 tem tempo de processamento 45 segundos. Esta comparação entre os tempos de processamento também se encontra na Tabela 5.3.

Comparando esses números, podemos ver que para nível máximo igual a 6, o processamento da construção regular é mais rápida mas, ao aumentar a resolução, a diferença entre os tempos de processamento dos dois tipos de construção acaba aumentando muito e a construção regular passa a ser mais lenta.

Observamos nos gráficos das Figuras 5.4 e 5.5, em processamentos seriais e paralelos, como o consumo de memória é alterado ao longo do tempo de execução do algoritmo. Como esperado, o pico de consumo de memória possui uma diferença de valor muito grande, dependendo se paralelizamos ou não o processamento de meta-células. Esse consumo se torna tão mais alto pois, além de precisarmos ter todas as meta-células em memória e não só apenas uma (o que já é oito vezes mais do que a inicialização de um processamento linear), cada uma das *threads* está processando e gerando mais dados.

Além disto, pelos gráficos podemos notar que o consumo de memória das construções adaptativas são nitidamente maiores que o consumo em construções regulares. O principal motivo por trás disto é o armazenamento de células de borda e suas células adjacentes. Ao processar uma meta-célula, precisamos continuar guardando para a etapa de conexão de meta-células os *voxels* contidos em células de borda e adjacentes. A construção adaptativa cria células em níveis menores de resolução, ou seja, células que representam espacialmente uma área maior do volume. Como consequência, estas células

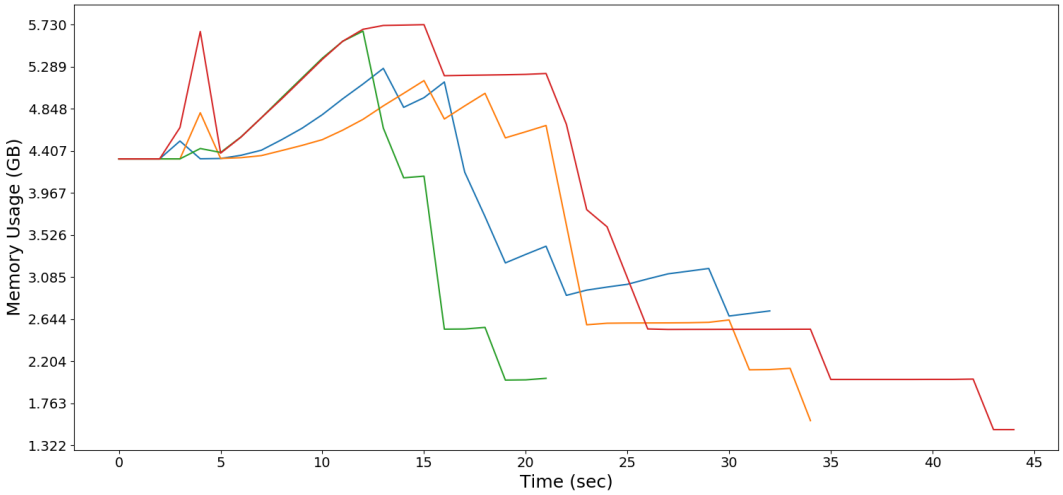

Figura 5.5: Consumo de memória dos processamentos paralelos ao longo do tempo. As curvas azul e amarela são referentes à construção adaptativa (níveis máximos 6 e 7, respectivamente) e, as curvas verde e vermelho, se referem à construção regular (também respectivos aos níveis indicados anteriormente).

devem guardar uma quantidade de *voxels* muito maior que as células que estão no nível máximo de resolução.

	Serial	Paralelo
Construção Regular (Nível 6)	129 segundos	23 segundos
Construção Regular (Nível 7)	226 segundos	78 segundos
Construção Adaptativa (Nível 6)	163 segundos	33 segundos
Construção Adaptativa (Nível 7)	183 segundos	45 segundos

Tabela 5.3: Comparação dos tempos de processamento entre construções regulares e adaptativas de forma serial e paralela. O nível entre parênteses indica o nível máximo da árvore da construção em questão.

5.4.2 Tamanho da superfície

Nesta seção comparamos brevemente as características espaciais das superfícies criadas. Como pode ser visto pela Tabela 5.4, ao usar uma construção regular, criamos a partir de uma *octree* de nível máximo 6 um total de 274.048 vértices e 137.024 triângulos, o que resulta em um arquivo de texto no formato OBJ de 8.6MB. Ao aumentar o nível de resolução, temos um aumento do número de vértices para 1.107.504, do número de faces para 553.752 e do tamanho do arquivo para 35MB.

Em uma construção adaptativa, em uma árvore de nível máximo 6 obtemos 87.138 vértices, 44.596 faces e um arquivo de saída de 2.7MB. Em

uma árvore de nível máximo 7 obtemos 302.839 vértices, 154.248 faces e um arquivo pesando 9.6MB.

Ao compararmos esses dois resultados, podemos concluir que o uso de uma construção regular resulta em um aumento significativo da quantidade de dados criados. O motivo dessa diferença está relacionado ao número de arestas processadas. Na construção regular de uma árvore no nível máximo 7, são processadas um total de 262.768 arestas enquanto na construção adaptativa de mesmo nível de resolução 68.384, quase quatro vezes menos arestas. Naturalmente, como consequência de um maior número de arestas processadas, temos um aumento de artefatos criados.

	Vértices	Polígonos	Tamanho
Construção Regular (Nível 6)	274.048	137.0234	8.6MB
Construção Regular (Nível 7)	1.107.504	553.752	35MB
Construção Adaptativa (Nível 6)	87.138	44.596	2.7MB
Construção Adaptativa (Nível 7)	302.839	154.248	9.6MB

Tabela 5.4: Comparação entre os arquivos de saída da isosuperfície extraída. O nível entre parênteses indica o nível máximo da árvore da construção em questão.

6
Conclusão e Trabalhos Futuros

Com nossa implementação, fomos capazes de atingir o objetivo inicial de criar uma representação poligonal de um domo sal segmentado dentro de um volume binário, combinando estratégias de extração de isosuperfícies com estratégias *out-of-core*. O método é capaz de criar um campo de distância de forma rápida a partir do volume binário e, a partir deste campo, criar uma superfície fechada sem buracos ou *cracks* utilizando dois métodos diferentes. A partir da construção adaptativa, podemos criar uma superfície que, além dos atributos já citados, também é livre de interseção entre seus polígonos. Utilizando uma construção regular, podemos criar uma superfície que possui uma qualidade visual melhor e também é *manifold*.

Como trabalhos futuros, pretendemos melhorar a eficiência da nossa solução, em termos de memória, de tempo e de qualidade da superfície. Embora o consumo de memória não seja tão alto, ainda há espaço para melhorias. Por exemplo, o cálculo do campo de distância depende da máquina possuir espaço em RAM suficiente para uma cópia do volume para ser realizada. Existem estratégias *out-of-core* que poderiam ser aplicadas neste caso, como a descrita em [Michikawa et al., 2007]. Embora estratégias *out-of-core* sejam mais lentas em processamento no geral, é importante para que máquinas com menor poder computacional possam fazer o pré-processamento sem depender da memória disponível.

Como dito na seção de resultados, para validar o funcionamento da estratégia de processamento paralelo das meta-células, apenas habilitamos a divisão do volume em oito meta-células. Embora se mostre suficiente pro dado que utilizamos de teste, para dados maiores seja talvez necessário permitir o aumento do número de meta-células disponíveis para uso.

Também queremos adicionar um critério geométrico de subdivisão de *octree*, que permita de fato criar uma superfície adaptativa. Este critério geométrico, como o descrito em [Lobello et al., 2014] e em [Lobello et al., 2012], nos permitiria refinar mais ou menos em áreas de maior curvatura, diminuindo mais a quantidade de dados de saída do algoritmo.

Por último, seria interessante poder criar uma superfície que tenha as características que as duas construções de *octree* proporcionam, ou seja, uma

superfície livre de interseções e ao mesmo tempo *manifold*. É possível criar uma superfície deste tipo utilizando a construção adaptativa quando todas as células folhas interceptadas não são complexas, porém, não é garantido que isto aconteça para qualquer nível de resolução definido.

Referências bibliográficas

[Aqrawi et al., 2011] AQRAWI, A. A.; BOE, T. H. ; BARROS, S.. **Detecting salt domes using a dip guided 3d sobel seismic attribute**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2011, p. 1014–1018. Society of Exploration Geophysicists, 2011.

[Di and AlRegib, 2017] DI, H.; ALREGIB, G.. **Seismic multi-attribute classification for salt boundary detection-a comparison**. In: 79TH EAGE CONFERENCE AND EXHIBITION 2017, volumen 2017, p. 1–5. European Association of Geoscientists & Engineers, 2017.

[Engel et al., 2004] ENGEL, K.; HADWIGER, M.; KNISS, J. M.; LEFOHN, A. E.; SALAMA, C. R. ; WEISKOPF, D.. **Real-time volume graphics**. In: ACM SIGGRAPH 2004 COURSE NOTES, p. 29–es. 2004.

[Felzenswalb et at., 2012] FELZENSZWALB, P. F.; HUTTENLOCHER, D. P.. **Distance transforms of sampled functions**. Theory of computing, 8(1):415–428, 2012.

[Garland and Heckbert, 1998] GARLAND, M.; HECKBERT, P. S.. **Simplifying surfaces with color and texture using quadric error metrics**. In: PROCEEDINGS VISUALIZATION'98 (CAT. NO. 98CB36276), p. 263–269. IEEE, 1998.

[Guennebaud et al., 2010] GUENNEBAUD, G.; JACOB, B. ; OTHERS. **Eigen**. URI: http://eigen. tuxfamily. org, 2010.

[Hoshen and Kopelman, 1976] HOSHEN, J.; KOPELMAN, R.. **Percolation and cluster distribution. i. cluster multiple labeling technique and critical concentration algorithm**. Physical Review B, 14(8):3438, 1976.

[Ju et al., 2002] JU, T.; LOSASSO, F.; SCHAEFER, S. ; WARREN, J.. **Dual contouring of hermite data**. In: PROCEEDINGS OF THE 29TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 339–346, 2002.

[Ju et al., 2006] JU, T.; UDESHI, T.. **Intersection-free contouring on an octree grid**. In: PROCEEDINGS OF PACIFIC GRAPHICS, volumen 2006. Citeseer, 2006.

[Kobbelt et al., 2001] KOBBELT, L. P.; BOTSCH, M.; SCHWANECKE, U. ; SEIDEL, H.-P.. **Feature sensitive surface extraction from volume data**. In: PROCEEDINGS OF THE 28TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, p. 57–66, 2001.

[Lewiner et al., 2010] LEWINER, T.; MELLO, V.; PEIXOTO, A.; PESCO, S. ; LOPES, H.. **Fast generation of pointerless octree duals**. In: COMPUTER GRAPHICS FORUM, volumen 29, p. 1661–1669. Wiley Online Library, 2010.

[Lobello et al., 2012] LOBELLO, R. U.; DUPONT, F. ; DENIS, F.. **Multi-resolution dual contouring from volumetric data.** In: GRAPP/IVAPP, p. 163–168, 2012.

[Lobello et al., 2014] LOBELLO, R. U.; DUPONT, F. ; DENIS, F.. **Out-of-core adaptive iso-surface extraction from binary volume data**. Graphical models, 76(6):593–608, 2014.

[Lobello et al., 2014] LOBELLO, R. U.; DENIS, F. ; DUPONT, F.. **Adaptive surface extraction from anisotropic volumetric data: contouring on generalized octrees**. Annals of Telecommunications-annales des télécommunications, 69(5-6):331–343, 2014.

[Lorensen et al., 1987] LORENSEN, W. E.; CLINE, H. E.. **Marching cubes: A high resolution 3d surface construction algorithm**. ACM siggraph computer graphics, 21(4):163–169, 1987.

[Meagher, 1982] MEAGHER, D.. **Geometric modeling using octree encoding**. Computer graphics and image processing, 19(2):129–147, 1982.

[Michikawa et al., 2007] MICHIKAWA, T.; TSUJI, K.; FUJIMORI, T. ; SUZUKI, H.. **Out-of-core distance transforms**. In: PROCEEDINGS OF THE 2007 ACM SYMPOSIUM ON SOLID AND PHYSICAL MODELING, p. 151–158, 2007.

[Moore, 1920] MOORE, E. H.. **On the reciprocal of the general algebraic matrix**. Bull. Am. Math. Soc., 26:394–395, 1920.

[Morton, 1966] MORTON, G. M.. **A computer oriented geodetic data base and a new technique in file sequencing**. 1966.

[Nielson, 2004] NIELSON, G. M.. **Dual marching cubes**. In: IEEE VISUALIZATION 2004, p. 489–496. IEEE, 2004.

[Satherley and Jones, 2001] SATHERLEY, R.; JONES, M. W.. **Vector-city vector distance transform**. Computer Vision and Image Understanding, 82(3):238–254, 2001.

[Schaefer and Warren, 2003] SCHAEFER, S.; WARREN, J.. **Dual contouring:" the secret sauce", rice university**. Technical report, Department of Computer Science Technical Report, 2003.

[Schaefer et al., 2007] SCHAEFER, S.; JU, T. ; WARREN, J.. **Manifold dual contouring**. IEEE Transactions on Visualization and Computer Graphics, 13(3):610–619, 2007.

[Shi et al., 2018] SHI, Y.; WU, X. ; FOMEL, S.. **Automatic salt-body classification using a deep convolutional neural network**. In: SEG TECHNICAL PROGRAM EXPANDED ABSTRACTS 2018, p. 1971–1975. Society of Exploration Geophysicists, 2018.

[Shi et al., 2019] SHI, Y.; WU, X. ; FOMEL, S.. **Saltseg: Automatic 3d salt segmentation using a deep convolutional neural network**. Interpretation, 7(3):SE113–SE122, 2019.

[Wenger, 2013] WENGER, R.. **Isosurfaces: geometry, topology, and algorithms**. CRC Press, 2013.