Pontifícia Universidade Católica do Rio de Janeiro

Campos de Distâncias em Superfícies Abertas

André Abreu Calfa

Projeto Final de Graduação

Centro Técnico Científico - CTC
Departamento de Informática
Curso de Graduação em Engenharia da Computação

Rio de Janeiro, Novembro 2013

André Abreu Calfa

Campos de Distâncias em Superfícies Abertas

Relatório de Projeto Final, apresentado ao curso Engenharia de Computação como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Waldemar Celes Filho

Rio de Janeiro,
Março 2013

Resumo

Campos de distâncias são vetores de duas ou três dimensões, cujo propósito é armazenar a menor distância entre um dado ponto e o objeto em questão. Além de calcular a menor distância, utilizando estes campos é possível também descobrir em qual direção a partir de um ponto do campo este objeto se encontra. Ademais, campos de distância têm mostrado como sendo de grande importância para o processamento de objetos em três dimensões. Este projeto procura atender à área de geologia, através do estudo do processamento em superfícies abertas, pois superfícies geológicas possuem tal característica.

Palavras-chave

campos de distância, superfícies abertas, triangulação.

Abstract

Distance fields are either two- or three-dimensional data arrays, which purpose is to store the closest distance from a given point to an object. In addition to calculating the minimum distance between a point and an object, by using distance fields it is also possible to calculate the direction to an element, from that given point. Distance fields have shown considerable importance in processing objects in 3D applications. This project will study an extension of this kind of processing in open surfaces, aiming to give support the geology areas, since geological surfaces have that specific characteristic.

Keywords

distance fields, open surfaces, triangulation.

ÍNDICE

1 Introdução

1.1 Motivação e Domínio do Sistema

Campos de distância possuem aplicações tanto em imagens de duas dimensões (2D) quanto em três dimensões (3D). Como um simples exemplo, estes campos servem como operadores que são aplicados à imagens binárias. Campos de distância foram citados pela primeira vez na literatura na década de 60, em um artigo sobre Processamento de Imagens por Rosenfeld e Pfaltz [Rosenfeld e Pfaltz 1966], onde o algoritmo de *Chamfer Distance Transform* é aplicado à uma imagem, além da representação mínima de um esqueleto em relação à sua estrutura original. A partir deste momento, vários autores encontraram melhorias na precisão deste algoritmo, assim como também introduziram algoritmos alternativos, por exemplo: *Vector Distance Transform*, *Fast Marching Methods*, e *Level Sets*.

Muitos dos primeiros trabalhos se limitavam ao processamento de imagens em duas dimensões. Entretanto, o fato de que conjuntos de dados no espaço tridimensional se tornaram importantes fez com que muitas pesquisas mais recentes voltassem para o processamento deste tipo de dado. Diversos domínios do mundo científico compartilham este estudo, sendo eles: a comunidade de Visão Computacional (Processamento de Imagem), a comunidade de Física (Frente de onda, soluções para a Equação Eikonal) e a comunidade de Computação Gráfica (representação de objetos e processamento). [Jones, Bæsrentzen e Sramek 2006]

Assim sendo, este projeto é uma extensão de um sistema já existente, visto que estudos em campos de distância já são feitos há algumas décadas. No entanto, a motivação é a de estudar soluções mais específicas para superfícies abertas, que compreendem superfícies geológicas. Neste projeto, campos de distância serão aplicados para o processamento das superfícies abertas. Através do cálculo de campos de distâncias, é possível reconstruir toda a superfície original, obtendo no final a forma definida da mesma, sem que ela precise estar em memória.

1.2 Definição do Problema

Um conjunto de dados D de um campo de distância que representa uma superfície S pode ser definido como: $D : \mathbb{R}^3 \to \mathbb{R}$ e para $p \in \mathbb{R}^3$,

$$D(p) = \cdot \min\{|p - q| : q \in S\} \tag{1}$$

onde | | é a norma Euclidiana.

O primeiro passo é então introduzir uma grade de três dimensões no espaço do problema. Cada vértice dessa grade, que é denominado de *voxel* (amostra do campo de distância) armazenará a sua respectiva distância mais curta para a superfície S em questão. Com isso, se quisermos calcular a distância de al-

gum ponto p' qualquer que não coincide com um dos *voxels* para objeto, basta realizarmos qualquer tipo de interpolação apropriada, por exemplo: interpolação trilinear.[Jones e Satherley 2001]

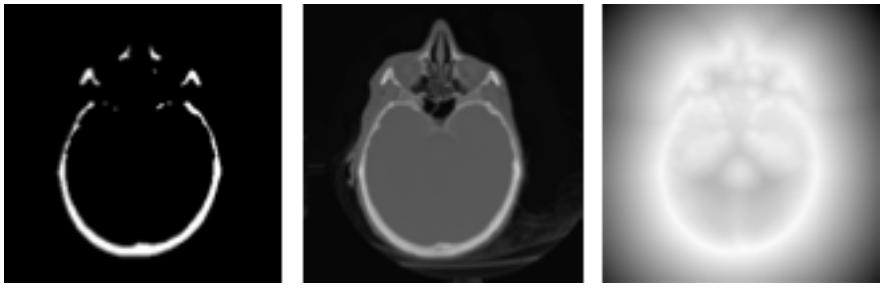

Figura 1: Da esquerda para direita: segmentação binária, imagem original e campo de distância de um crânio.

1.3 Justificativa da Relevância do Problema

Como já mencionado anteriormente, campos de distância possuem diversas aplicações em diferentes áreas da ciência. No caso específico deste projeto, o desenvolvimento de sistemas computacionais nas áreas de Geologia Estrutural, em particular nas áreas de Estratigrafia, Geotectônica e Sedimentologia, tem sido de grande importância para que a exploração de petróleo no Brasil seja possível. Logo, sistemas que provêem assistência à interpretação de modelos geológicos realizam processos de sedimentação de bacias geológicas e movimentação das placas tectônicas em duas e três dimensões permitem ao geólogo ou geofísico aprimorar substancialmente a interpretação estrutural de bacias geológicas.

Em síntese, estes sistemas colaboram no processo de detecção de alguns elementos e fenômenos geológicos, tornando-se indicadores da atratividade das regiões em observação para a prospecção petrolífera, cooperando para uma redução do risco exploratório.[*Modelagem Geológica de Sistemas Petrolíferos*]

Para ilustrar melhor as tarefas realizadas por um geólogo que implicam uma cooperação direta com programas de computador:

- Definição da área de trabalho ou projeto.
- Escolha dos poços e tipos de poços a usar.
- Escolha da malha sísmica a ser apresentada.
- Desenho do traçado das seções desejadas.
- Interpretação integrada das seções.
- Geração de mapas e superfícies integradas às seções.

1.4 Ambiente Computacional

O ambiente tecnológico de desenvolvimento utilizado neste projeto final é caracterizado pela utilização da plataforma computacional *Macintosh*, contendo o sistema operacional na versão Mac OS X 10.8. As especificações de *hardware* encontram-se na tabela 1.

Nome do Processador	Intel Core i7
Velocidade do Processador	2.66 GHz
Número de Processadores	1
Número Total de Núcleos	2
L2 Cache (por Núcleo)	256 KB
L3 Cache	4 MB
Memória	8 GB DDR3 1067MHz
Compilador	GCC 4.8.2

Tabela 1: Configurações de Sistema

Este sistema descrito acima é um *notebook* da *Apple Macintosh*. A escolha deste *hardware* tem como justificativa a fácil portabilidade do mesmo entre diferentes localidades em que trabalhou-se neste projeto, sendo eles minha própria residência e o campus da PUC-Rio, aliada a grande performance computacional que pode ser atingida por esta máquina.

Como plataforma de desenvolvimento do *software*, a linguagem de programação escolhida foi C++, pelo fato de que esta linguagem possui excelente desempenho geral já que faz uso de ponteiros que fornecem rápido acesso à dados, além de estar entre as linguagens de programação mais utilizadas em todo o mundo.

O utilitário para compilar automaticamente programas e bibliotecas adotado para este projeto foi o Tecmake, ferramenta desenvolvida pelo Tecgraf. Ela é essencialmente um *GNU Makefile* com uma série de *targets* predefinidos que possui o objetivo de oferecer as ferramentas necessárias para o desenvolvimento de bibliotecas e aplicações, para que versões de diferentes plataformas possam ser geradas de maneira homogênea. Em outras palavras, o usuário do Tecmake não precisa se preocupar com qual é a plataforma sendo usada no momento em que ele cria seus códigos fonte. O Tecmake possui licença de *software* baseada na licença MIT, o que significa que esta ferramenta é *software* livre e pode ser usada tanto para razões acadêmicas ou comerciais, sem custo algum. [*Tecmake Build Tool*]

2 Estado da Arte

2.1 Descrição e Avaliação de Tecnologias e Sistemas Existentes

A realização deste projeto teve como base o estudo de alguns artigos, trabalhos e relatórios técnicos. Os artigos [Jones e Satherley 2001] e [Jones, Bæsrentzen e Sramek 2006] apresentam uma introdução à técnica de campos de distância, definindo conceitos como o de que cada ponto destes campos representam a menor distância do mesmo até qualquer objeto do domínio em questão. Ademais, outras propriedades igualmente importantes podem ser auferidas, como qual direção para a superfície e, em caso do campo de distância possuir sinal, descobrir se um ponto está dentro ou fora da superfície (no caso da superfície possuir volume). Ademais, os autores demonstram aplicações nas áreas de visão computacional, física e computação gráfica, assim como diferentes métodos (algoritmos) que podem ser implementados para calcular campos de distância, comparando suas complexidades e quão simples eles são de implementar.

O relatório técnico feito por [Perry e Frisken 2001] foi de suma importância para a criação da biblioteca proposta neste projeto final, que realiza a reconstrução de superfícies triangulares a partir de campos de distância, usando um algoritmo de triangulação. O sistema desenvolvido por [Perry e Frisken 2001], Kizamu, é capaz de criar personagens digitais para a indústria de entretenimento, incorporando uma variados algoritmos e avanços tecnológicos, além de novos paradigmas de interação. No núcleo de seu sistema, Kizamu utiliza ADFs (campos de distância adaptativos, sigla em inglês).

Por fim, o artigo escrito por [Vanco e Kreylos 2011] demonstra uma aplicação de campos de distância em geologia. A forma de placas tectônicas que se encontram no manto da Terra são cruciais para vários estudos em geociência, como a definição de forças que realizam o movimento das placas tectônicas, o potencial sísmico e os riscos de tsunami. A modelagem de deformações, assim como os parâmetros de entrada, como por exemplo: temperatura, viscosidade e geometria são realizados através de elementos finitos, para garantir a construção de um campo de distância suave e contínuo em torno da placa tectônica. Este trabalho utiliza um *framework* para processar dados sísmicos (localização de terremotos) e define a forma da placa tectônica, além de computar campos de distância contínuos em uma malha com espaçamento variado de nós. O campo de distância resultante então reúne distâncias até a aproximação contínua da placa e distâncias até a fronteira da placa.

2.2 Algoritmo Força Bruta

Uma solução para calcular o campo de distância é a utilização do algoritmo de força bruta. Seu cálculo se resume a primeiramente estabelecer, para cada *voxel* v, seu campo de distância com o valor ∞. Em seguida, calcula-se a menor

de distância de v para superfície S, através do cálculo individual da distância entre v e cada triângulo da superfície, armazenando-se sempre a distância mínima de v até S.

Este algoritmo em teoria é capaz de produzir campos de distância com precisão no sentido de que cada ponto da grade possui uma distância precisa até a superfície, codificada pela função de segmentação. A complexidade deste método é de $O(m \cdot n)$, onde m é número de voxels do campo de distância e n é o número de triângulos da superfície.

3 Objetivos do Trabalho

3.1 Biblioteca ReMGeo

O escopo deste projeto tem como objetivo implementar uma biblioteca que reconstrua superfícies geológicas (triangulares), a partir de campos de distância. A biblioteca foi nomeada ReMGeo (Reconstrutor de Malhas Geológicas) e tem como entrada uma superfície triangular, e como saída a superfície reconstruída após calcular o campo de distância. Além disso, construir uma aplicação que permite a visualização tridimensional dos resultados também faz parte do projeto.

3.2 Escopo do Sistema

Tendo em vista os objetivos citados acima e as informações contidas na seção 1.3, o público alvo para este projeto consiste nos engenheiros e geólogos que trabalham com modelagem geológica, em especial o laboratório Tecgraf, que em conjunto com a Petrobras visam otimizar a etapa de exploração de petróleo, garantindo menor risco exploratório.

4 Atividades Realizadas

4.1 Estudos Preliminares

Ao iniciar este projeto, precisei obter conhecimento sobre campos de distância e suas aplicações. Previamente, o recursos básicos de álgebra linear e cálculo foram adquiridos através de cursos do Departamento de Matemática da PUC-Rio. Ademais, por estar cursando Engenharia de Computação nesta universidade foi possível agregar um considerável conhecimento em diferentes áreas da Ciência da Computação, como Algoritmos, Engenharia de Software, Computação Gráfica e outras disciplinas.

4.2 Estudos Conceituais e de Tecnologia

Para realizar este projeto, será necessário primeiramente revisar diversos tópicos em Álgebra Linear, como por exemplo vetores, retas, planos e espaços Euclidianos. Além disso, estudos serão feitos para aprofundar o conhecimento em campos de distância e seus diversos algoritmos que buscam implementar uma solução ótima no cálculo destes campos.

Em segundo lugar, algumas bibliotecas, ferramentas e variáveis de ambiente do Tecgraf serão utilizadas para auxílio neste projeto. Portanto, é essencial que se estude o funcionamento de cada um desses itens citados anteriormente para que não haja uso incorreto dos mesmos ou falhas na implementação e execução deste projeto.

4.3 Cronogramas

4.3.1 Cronograma para Desenvolvimento Projeto Final I

- Estudo através da leitura de artigos e da análise da implementação destes trabalhos;
- Desenvolvimento de um código que calcula a menor distância de um ponto p à um triângulo T, a partir da projeção de p no plano em que T se encontra. [Eberly 2008]
- Elaborar um programa de teste que verifica a corretude dos cálculos de distância do item anterior.
- Desenvolvimento de um algoritmo que calcula a menor distância de um ponto p à uma malha de triângulos M, aplicando o código elaborado no item anterior para todos os triângulos em M.
- Desenvolvimento de um algoritmo que calcula a menor distância de cada ponto p contido em uma grade G à uma malha de triângulos M.
- Criar uma unidade de teste que verifica a corretude do cálculo das distância descrita no item anterior, comparando o resultado da distância obtido pelo algoritmo para diversos pontos, com a distância euclidiana dos mesmos pontos para um triângulo determinado *a priori*.

4.3.2 Cronograma para Desenvolvimento Projeto Final II

- Implementar uma visualização 3D da malha de triângulos.
- Criar a estrutura de grade de pontos e aplicar o cálculo de distância de cada ponto da grade para a malha de triângulos.
- Acrescentar a grade de pontos à visualização 3D.
- Calcular o sinal do campo de distância e visualizá-lo simultaneamente com a malha de triângulos.
- Adicionar técnicas de paralelização no cálculo da distância visando melhor

performance do mesmo.

- Implementar algoritmo de reconstrução da malha de triângulos original a partir do campo de distâncias.
- Desenvolver algoritmo que evite o serrilhamento das bordas da superfície em sua reconstrução.

5 Computação de Campos de Distância

5.1 Representação de Superfícies

A representação de superfícies feita por malha de triângulos é uma das mais utilizadas em representações de geometrias tridimensionais. Por isso, é especificamente crucial que se obtenha o campo de distâncias da malha de triângulo, sem considerar o sinal na reconstrução. Isto se deve pelo fato de que esta informação só é relevante em superfícies fechadas, onde seria necessário distinguir entre pontos do campo que estão fora da superfície de pontos que estão dentro da superfície. Só é possível gerar campos de distância a partir de uma determinada classe de malha de triângulos: as com variedade de dimensão 2 (2-*manifold*) e orientáveis. A condição de variedade pode ser obtida através de: [Hoffmann 1989]

1. A malha não deve conter quaisquer auto interseções: Seus triângulos só podem compartilhar arestas e vértices e precisam ser disjuntos.
2. Cada aresta deve ser adjacente a somente dois triângulos.
3. Triângulos incidentes em um vértice devem formar um único ciclo em torno de cada vértice.

Se as características da malha obedecem a todas as condições, ela particiona bem definidamente o espaço em regiões internas, externas e a própria malha.

Em termos de programação, a estrutura de dados utilizada para representar uma malha de triângulos é a biblioteca CSI (*Cross-Section Interpreter*). Esta biblioteca faz parte de um projeto iniciado em 1998, e é amplamente empregada em aplicações desenvolvidas pelo laboratório Tecgraf, como por exemplo: o Interage e o Recon. A CSI possui um conjunto de classes, onde cada classe representa um tipo de entidade geológica, seguindo a linha de *"design pattern"*: listas, vetores, fábricas, tabelas *hash* e etc.

As superfícies são carregadas em memória através da leitura de um arquivo no formato GOCAD [*Gocad*]. Este arquivo contém a relação de todos os triângulos da malha e cada um de seus três vértices. A classe responsável para importar e exportar arquivos GOCAD é a classe CsiTSurf (a extensão do arquivo é ".ts"). A malha representada pela estrutura dessa classe possui um sistema de *Undo/Redo* interno. Isto significa que qualquer ação aplicada em uma malha CSI por um dos métodos implementados em sua classe produz uma opção de *"undo"*, que por sua vez, ao ser realizada, produz uma alternativa de *"redo"*.

5.2 Distância de um Ponto à um Triângulo

5.2.1 Formulação Matemática

Para se calcular a distância de um ponto à um triângulo são necessárias algumas formulações. Primeiramente, tomamos um ponto P e um triângulo parametrizado $T(s,t) = B + sE_0 + tE_1$ para $(s,t) \in D = \{(s,t) : s \in [0,1], t \in [0,1], s+t \leq 1\}$. A menor distância pode ser computada ao se encontrar os valores $(\bar{s},\bar{t}) \in D$ que correspondem ao ponto contido em T mais próximo à P.

A função que define a distância quadrática de qualquer ponto do triângulo para P é: $Q(s,t) = |T(s,t) - P|^2$ para $(s,t) \in D$. Expandindo esta função, temos:

$$Q(s,t) = as^2 + 2bst + ct^2 + 2ds + 2et + f, \tag{2}$$

onde $a = E_0 \cdot E_0$, $b = E_0 \cdot E_1$, $c = E_1 \cdot E_1$, $d = E_0 \cdot (B-P)$, $e = E_1 \cdot (B-P)$, e por fim $f = (B-P) \cdot (B-P)$. É importante lembrar que funções quadráticas são classificadas pelo sinal de $ac - b^2$. No caso da função $Q(s,t)$,

$$\begin{aligned} ac - b^2 &= (E_0 \cdot E_0)(E_1 \cdot E_1) - (E_0 \cdot E_1)^2 \\ &= |E_0 \times E_1|^2 > 0 \end{aligned} \tag{3}$$

Pelo fato de que as arestas E_0 e E_1 do triângulo $T(s,t)$ são linearmente independentes, o produto vetorial entre elas é um vetor não nulo e consequentemente positivo por ser elevado à potência de 2. Portanto, fica claro definir que o gráfico de $Q(s,t)$ é um parabolóide.

O objetivo então é minimizar $Q(s,t)$ no domínio D. Nota-se que a função Q é continuamente diferenciável, logo o mínimo está localizado ou em um ponto no interior de D, onde o gradiente $\nabla Q = 2(as + bt + d, bs + ct + e) = (0,0)$ ou em um ponto na fronteira de D.

5.2.2 O Algoritmo

Para minimizar Q, precisamos encontrar os valores de s e t que tornam ∇Q nulo. Os valores são: $\bar{s} = (be - cd)/(ac - b^2)$ e $\bar{t} = (bd - ae)/(ac - b^2)$. Como mencionado anteriormente, caso $(\bar{s},\bar{t}) \in D$, o mínimo de Q já está definido. Caso contrário, o mínimo deve estar contido na fronteira do triângulo. Para melhor visualização das fronteiras, consideremos a figura 2:

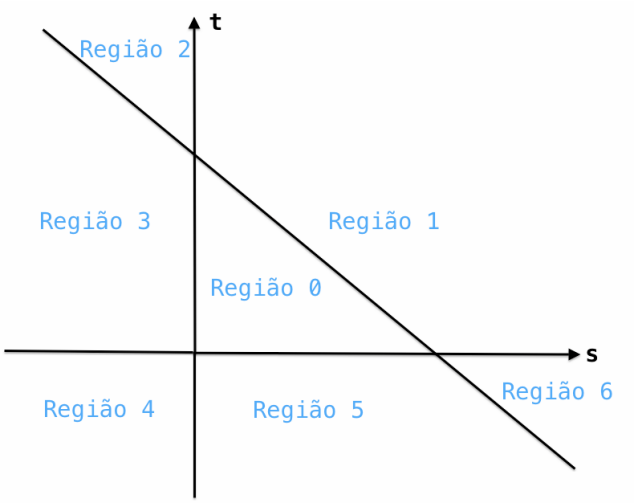

Figura 2: Partição do plano st através do triângulo $T(s,t)$ no domínio D.

Analisemos agora as diferentes regiões em que $(\bar{s},\bar{t})$ pode estar localizado:

Região 0

O triângulo localizado no centro da figura, denominado de região 0, corresponde ao domínio de Q, $(s,t) \in D$. Este é um caso trivial pois o ponto do triângulo mais próximo de P é interior ao triângulo.

Regiões 1, 3, 5

Se $(\bar{s},\bar{t})$ está na região 1, para facilitar o estudo deste caso podemos analisar as curvas de nível de Q no plano st em que Q é constante. A figura 3 ilustra a situação descrita. Já foi dito anteriormente que o gráfico de Q consiste em um parabolóide, e por isso as curvas de nível são elipses. A curva de nível se degenera à um único ponto $(\bar{s},\bar{t})$ quando $\nabla Q = (0,0)$. O mínimo global ocorre neste ponto, rotulado V_{min}. Os valores de nível V aumentam em relação à V_{min} quanto mais afastados de $(\bar{s},\bar{t})$, correspondendo às elipses mais distantes do mesmo. Conforme se incrementa o valor de nível, será possível atingir o valor mínimo V_0, cuja elipse (implicitamente definida como $Q = V_0$) apenas toca a aresta $s + t = 1$ do triângulo, onde $s = s_0 \in [0,1]$, $t_0 = 1 - s_0$. Sendo assim, para qualquer valor $V < V_0$, a elipse correspondente não intercepta o domínio D. Analogamente, para valores $V > V_0$, parte da elipse se encontra dentro de D. O objetivo então é encontrar o ponto (s_0,t_0), que provê a distância euclidiana quadrática mínima entre P e o triângulo T.

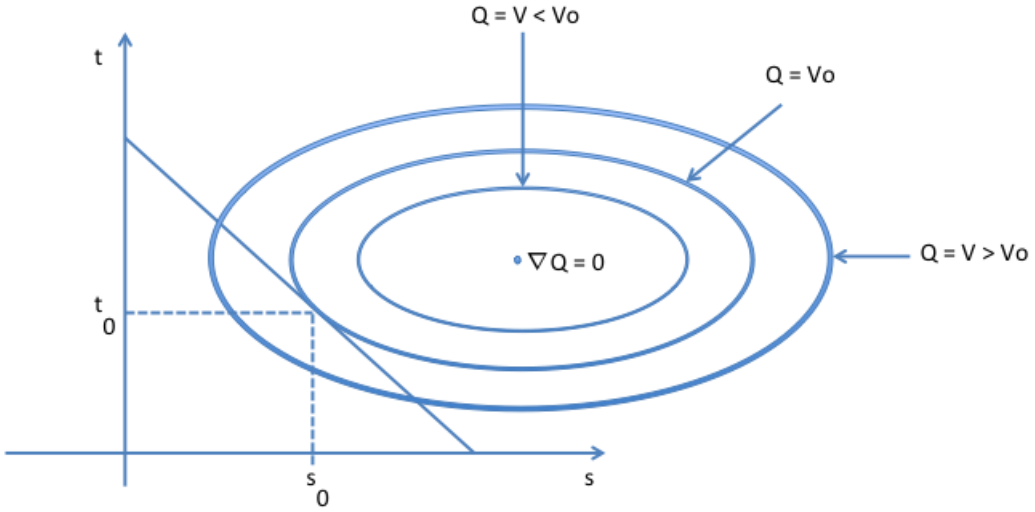

Figura 3: Diversas curvas de nível $Q(s,t) = V$.

Para facilitar o problema, podemos cortar o gráfico de Q com o plano $s + t = 1$ para visualizar melhor onde está o ponto na fronteira do triângulo de menor distância para P. Deste modo a curva de interseção é uma parábola descrita por $F(s) = Q(s, 1 - s)$ para $s \in [0, 1]$. Agora o mesmo problema foi reduzido para apenas uma dimensão, sendo que a meta é minimizar a função $F(s)$ onde $s \in [0, 1]$. O mínimo de $F(s)$ deve ocorrer ou ao longo da aresta (ponto interior de $[0, 1]$, onde $F'(s) = 0$ neste ponto), ou em uma das extremidades (ponto $s = 0$ ou $s = 1$). No caso da figura 3 é mostrado o caso em que o mínimo está localizado num ponto interior. Neste ponto a elipse tangencia a linha $s + t = 1$.

Resta agora saber diferenciar entre ponto interior e as extremidades. Para isso podemos também particionar o espaço unidimensional. O intervalo $[0, 1]$ divide a linha $s + t = 1$ em três intervalos: $s < 0$, $s \in [0, 1]$, e $s > 1$. Considere $F'(\hat{s}) = 0$. Se $\hat{s} < 0$, então $F(s)$ é uma função crescente para $s \in [0, 1]$. Logo, o mínimo deste intervalo deve estar em $s = 0$, em que no caso Q atinge seu mínimo em $(s, t) = (0, 1)$. Analogamente, se $\hat{s} > 1$, então $F(s)$ é uma função decrescente no mesmo intervalo $[0, 1]$. O mínimo de F então ocorrerá em $s = 1$ e o mínimo para Q está localizado em $(s, t) = (1, 0)$. Por fim, se $\hat{s}$ está contido dentro do intervalo $[0, 1]$, F possui seu mínimo em $\hat{s}$, e Q em $(s, t) = (\hat{s}, 1 - \hat{s})$.

O problema é resolvido similarmente caso $(\bar{s}, \bar{t})$ se encontre nas regiões 3 ou 5. Caso $(\bar{s}, \bar{t})$ esteja localizado na região 3, o mínimo ocorre em $(0, t_0)$ para $t_0 \in [0, 1]$. Se $(\bar{s}, \bar{t})$ está na região 5, então o mínimo se encontra em $(s_0, 0)$ onde $s_0 \in [0, 1]$.

Regiões 2, 4, 6

Se $(\bar{s}, \bar{t})$ está na região 2, é possível que a curva de nível Q que exerce o primeiro contato com o triângulo toque ou a aresta $s + t = 1$ ou a aresta $s = 0$. Para

determinar qual o caso, tomemos o seguinte fato: como o mínimo global ocorre na região 2, o negativo do gradiente da curva Q nunca poderá apontar para dentro de D. Logo, existe uma curva de nível Q que toca o triângulo cuja região interna não se sobrepõe com a região interna do triângulo. Esta curva de nível realiza uma das seguintes alternativas:

- Toca o vértice $(0, 1)$.
- Toca a aresta $s = 0$.
- Toca a aresta $s + t = 1$.

  A figura 4 ilustra este caso.

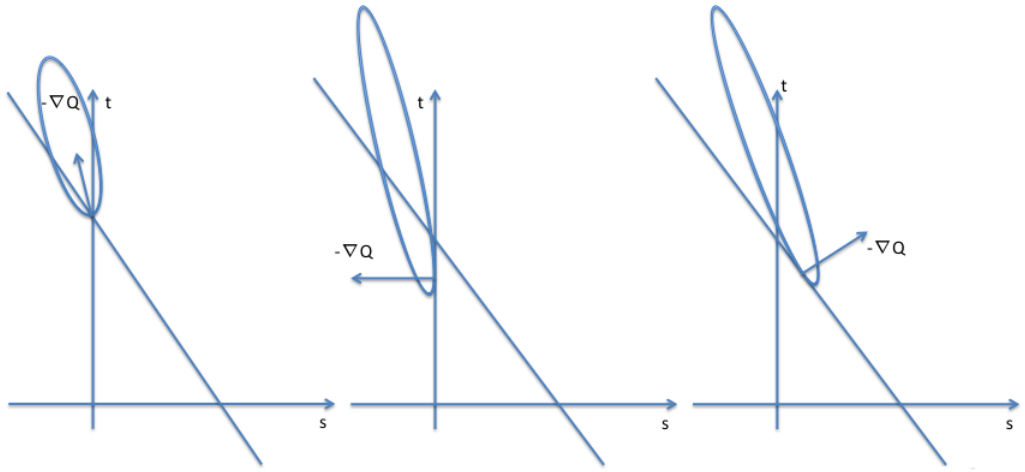

Figura 4: Três diferentes maneiras em que a curva de nível centrada na região 2 toca o triângulo.

Para qualquer ponto da curva de nível, $-\nabla Q$ aponta para o sentido de dentro da curva, o que significa que este gradiente não pode apontar para dentro do triângulo. Para descobrir qual dos três casos apresentados na figura acima é aquele em questão, se $\nabla Q = (Q_s, Q_t)$ onde Q_s e Q_t são derivadas parciais de Q, é necessário que $(0, -1) \cdot \nabla Q(0, 1)$ e $(1, -1) \cdot \nabla Q(0, 1)$ não podem ser ambos negativos. Os dois vetores $(0, -1)$ e $(1, -1)$ são as direções das arestas $s = 0$ e $s+t = 1$, respectivamente. A escolha da aresta correta é feita de acordo com qual dos produtos internos resultou em um valor negativo. O mesmo tipo de estratégia é aplicado para as regiões 4 e 6. Para a região 4, as duas parcelas cujos sinais devem ser analisados são $(1, 0) \cdot \nabla Q(0, 0)$ e $(0, 1) \cdot \nabla Q(0, 0)$.

5.2.3 Considerações sobre a Implementação

Em teoria, o mínimo global é encontrado através do cálculo de $\bar{s} = (be-cd)/\delta$ e $\bar{t} = (bd - ae)/\delta$ para que $\nabla Q(\bar{s}, \bar{t}) = (0, 0)$. A localização do mínimo global é averiguada para determinar se ele está no domínio D do triângulo. Se este for o caso, então já possuímos todas as informações necessárias para computar a distância mínima. Caso contrário, a fronteira de D precisa ser testada. Para postergar a divisão por δ, a implementação apenas calcula $\bar{s} = be-cd$ e $\bar{t} = bd-ae$

e verifica se o mínimo se encontra dentro do domínio com a seguinte escala: $s \in [0, \delta]$, $t \in [0, \delta]$ e $s + t \leq \delta$. A divisão é feita caso ele esteja neste conjunto. Caso contrário, a fronteira do triângulo é testada.

5.3 Construção da Nuvem de Pontos

5.3.1 Posicionamento dos Pontos e Cálculo da Distância

O próximo passo para calcular o campo de distância de uma superfície é a definição dos pontos que constituem a grade de pontos. A ideia é englobar a superfície com uma caixa envolvente (*bounding box*). Para isso, só é preciso conhecer as coordenadas x, y e z mínimas e máximas da superfície. Os pontos da grade estão contidos ao longo da caixa envolvente, espaçados por uma distância constante em cada uma das três direções. Para facilitar a visualização desta grade, a figura 5 representa uma grade em três dimensões de uma superfície.

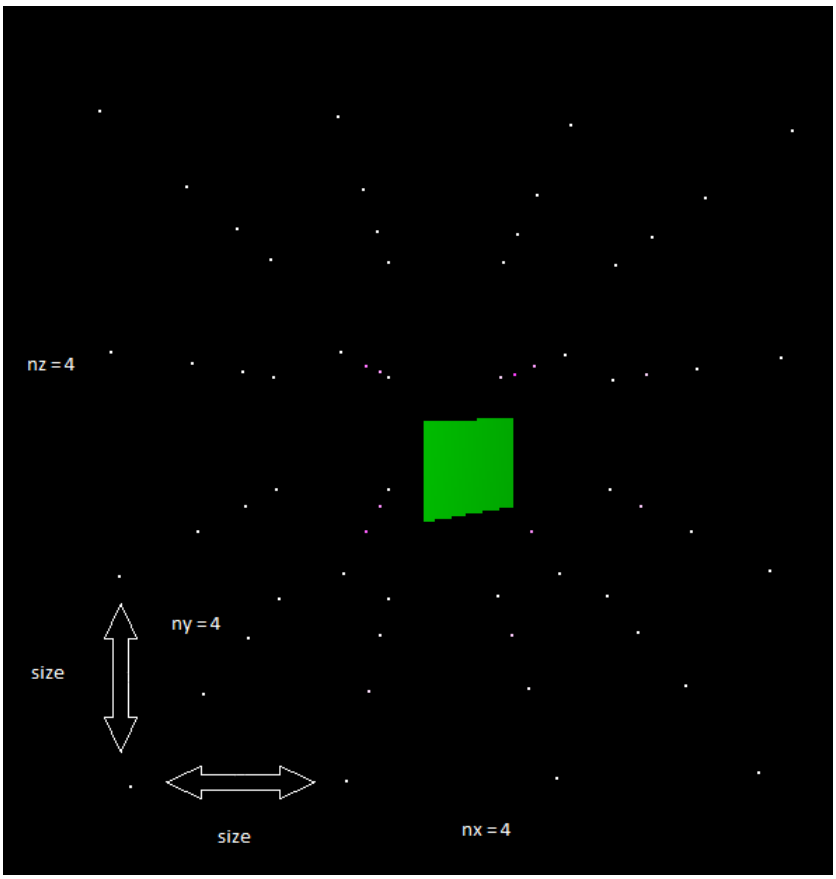

Figura 5: Ilustração da superfície (em verde) e os pontos da grade (em branco).

Diferentes superfícies possuem grandes variações em suas dimensões, em ambos os sentidos x, y, e z. Por isso, é importante que o valor da distância constante mencionada no parágrafo anterior (e denominada *size* na figura 5) seja válido. Um algoritmo que calcula valores aceitáveis para este espaçamento encontra-se a seguir.

Algorithm CalculaEspacamento( ):

$min \leftarrow 0$
for all triângulo t da malha de triângulos **do**
$minXYZ \leftarrow$ Menor coordenada dos três vértices de t, em cada dimensão
$maxXYZ \leftarrow$ Maior coordenada dos três vértices de t, em cada dimensão
$dist \leftarrow$ Soma das diferenças entre as coordenadas máximas e mínimas dos três vértices de t
if $dist < min$ **then**
$min \leftarrow dist$
end if
end for
return min

Uma vez que a funcionalidade de cálculo de distância de um ponto a um triângulo está implementada, basta replicar este mesmo cálculo para cada ponto da grade. Considerando o algoritmo de força bruta. Cada ponto do grid visita todos os triângulos da superfície e armazena a distância mínima do mesmo até a malha. O pseudocódigo deste algoritmo encontra-se abaixo:

Algoritmo DistanciaGradeMalha( ):

for all ponto i da grade de pontos **do**
$DistanciaMinima[i] \leftarrow \infty$
for all triângulo t da malha de triângulos **do**
$DistanciaCorrente \leftarrow DistanciaPontoTriangulo(i, t)$
if $DistanciaCorrente < DistanciaMinima[i]$ **then**
$DistanciaMinima[i] \leftarrow DistanciaCorrente$
end if
end for
end for

6 Reconstrução da Malha a partir do Campo de Distância

Uma interessante propriedade dos campos de distância é a possibilidade de se reconstruir a malha de triângulos original, através de um método de triangulação, fazendo-se uso apenas dos dados de distância e do gradiente da função de distância em cada ponto do campo. Para isso, precisa-se analisar as células (conjunto de oito vértices do campo de distância que formam um cubo) da grade de pontos. Os gradientes são relevantes neste processo porque caso em uma mesma célula existam gradientes apontando em sentidos opostos, isto significa que parte da superfície se encontra dentro dos limites desta célula, e com isso esta célula é necessária para a restauração da superfície original. Os passos deste algoritmo de reconstrução por triangulação encontram-se a seguir:

- Adiciona-se um vértice no centro de cada célula do *grid* em que a superfície original corta.
- Este mesmo vértice é ligado com outros dois vértices de células vizinhas para formar triângulos, a partir da seguinte regra: um triângulo une vértices de três células vizinhas que compartilham uma aresta em comum (e por isso o triângulo está associado com as arestas das células). Esta associação é feita somente se há *zero crossing* do campo de distância, isto é, a superfície a ser reconstruída precisa intersectar esta aresta. Pelo fato das células possuírem doze arestas, cada célula pode ter no máximo doze triângulos associados à ela. Para evitar a criação de triângulos redundantes, consideraremos somente seis das doze arestas possíveis para cada célula. Duas diferentes propostas encontram-se nas seções 6.1 e 6.2.
- Após todos os triângulos estarem formados e possuirmos uma malha topologicamente consistente, precisamos mover os vértices desta malha em direção à superfície original, ou seja, na direção do gradiente do campo distância, com um passo de valor igual ao da distância deste mesmo vértice até a superfície. Esta etapa está descrita na seção 6.3.

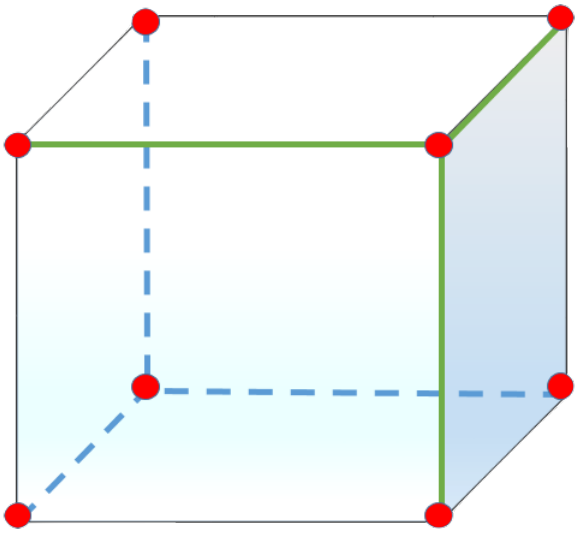

Figura 6: Célula da grade de pontos. As seis arestas consideradas para a construção de triângulos são as de espessura mais grossa, nas cores azul e verde. Os pontos vermelhos são os *voxels* do campo de distância.

Como mencionado anteriormente, apenas seis das doze arestas de uma célula são levadas em consideração na reconstrução da malha original para evitar redundâncias. A figura 6 ilustra estas seis arestas. A tabela abaixo especifica cada uma destas arestas e a ordem com que elas devem ser consideradas para garantir uma orientação consistente dos triângulos a serem formados. [Perry e Frisken 2001]

Aresta	**Face 1**	**Face 2**
superior direita	superior	direita
inferior esquerda	inferior	esquerda
superior frontal	superior	frontal
inferior traseira	inferior	traseira
frontal direita (vertical)	frontal	direita
traseira esquerda (vertical)	traseira	esquerda

A principal questão neste ponto do algoritmo é: como determinar quais células são intersectadas pela superfície original? A seguir estão descritas duas aproximações para encontrar a resposta deste problema.

6.1 Primeira Tentativa: Sinais dos *voxels*

6.1.1 Cálculo do Sinal do Campo de Distância

Podemos convencionar que pontos do *grid* que se encontram de um lado da superfície receberão sinal positivo na distância, enquanto os outros pontos que se localizam no lado oposto da superfície ficarão com sinal negativo na distância.

Uma forma de calcular o sinal do campo de distância consiste no seguinte cálculo algébrico: após calcular a menor distância do ponto da grade até a superfície, computar o produto interno entre o vetor normal do triângulo da malha que se encontra mais próximo ao ponto, com o vetor formado pelo próprio ponto e o vértice do triângulo. Deste modo é possível diferenciar pontos que estejam de um lado ou de outro da superfície pois, para todos os pontos em que o ângulo entre os dois vetores citados anteriormente for maior que $90°$, implica em resultado com sinal negativo e com isto eles estão do mesmo lado. Analogamente para o caso onde o produto interno resulta em um valor positivo, onde o ângulo entre os vetores é menor que $90°$. A figura abaixo demonstra este processo.

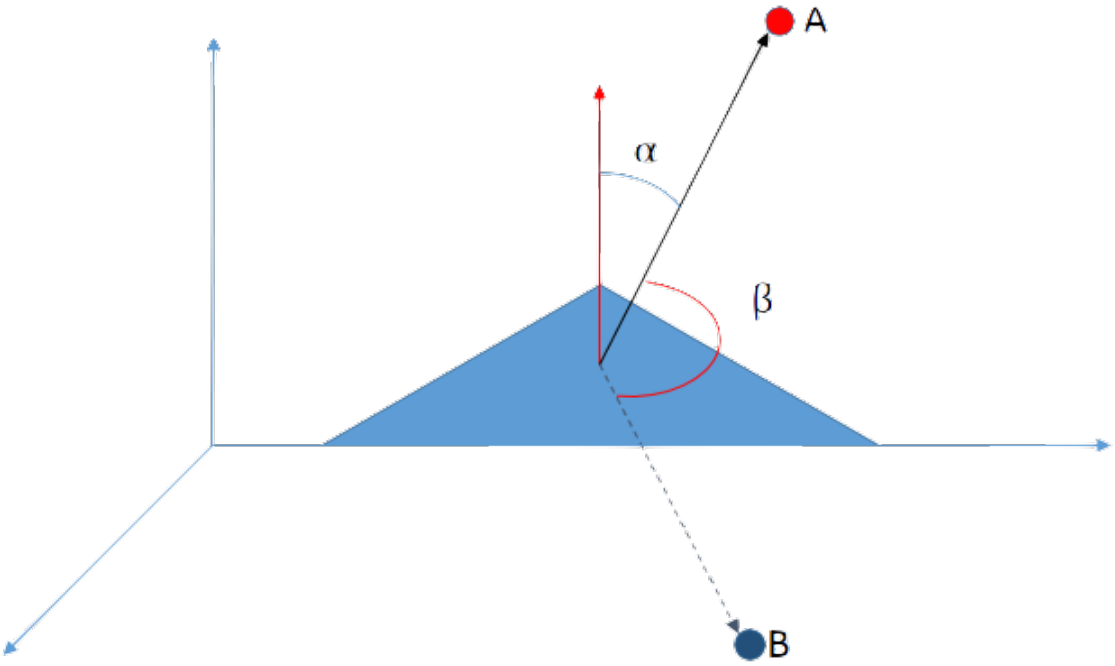

Figura 7: Os pontos A e B estão de lados opostos do triângulo. Por isso $\alpha < 90°$, $\beta > 90°$ e o produto interno de cada um dos vetores demonstrados resultará em sinais também opostos. O vetor em vermelho representa a normal do triângulo.

É crucial atentar para o fato de que qualquer que seja a orientação utilizada para o vetor da normal, cada triângulo da superfície precisa conter a mesma orientação. Orientações diferentes para cada triângulo torna este algoritmo falho. Logo, para garantir que os vetores sigam sempre a mesma orientação basta que para cada dois triângulos adjacentes, suas arestas em comum precisam ter orientações opostas. Por exemplo, se num primeiro triângulo a aresta $\bar{AB}$ é considerada para o cálculo da normal, no segundo triângulo adjacente precisa tomar a aresta $\bar{BA}$ para o mesmo cálculo.

6.1.2 Resultado

A primeira tentativa proposta é a de analisar os sinais de cada *voxel* de uma célula. Este método, em um rápido diagnóstico, aparenta ser efetivo: caso em uma mesma célula existam pontos do campo de distância cujos valores possuem sinais positivos e negativos, significa que tanto há pontos da célula que se localizam de um lado da superfície, quanto outros pontos da mesma célula que se encontram do outro lado da superfície. Entretanto, esta aproximação para o problema não é válida para todos os casos. A figura 8 mostra a razão: a superfície, ilustrada em duas dimensões, está representada pela linha de cor azul e os pontos em vermelho constituem pontos do campo de distância cujos valores têm sinal positivo,

enquanto os *voxels* em azul compõem *voxels* do campo de distância com valor negativo.

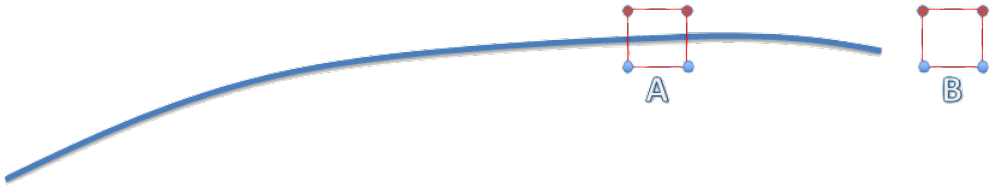

Figura 8: Caso onde a avaliação dos sinais de cada *voxel* da célula é falha. Na célula A, foi corretamente constatado que há interseção entre a superfície e a célula, entretanto a célula B também apontaria uma interseção já que possui pontos com valores positivos e negativos, porém visivelmente não há interseção nesta célula.

Logo, apesar de ser adequado em alguns casos, este método não pode ser utilizado pois ele não é capaz de discernir entre células que realmente cortam a superfície e células que, mesmo possuindo pontos com diferentes sinais no valor de distância, não se encontram dentro dos limites da malha de triângulos.

6.2 Segunda Tentativa: Gradiente dos *voxels*

Uma estratégia alternativa para solucionar esta questão é calcular o gradiente do campo de distância para cada *voxel* do grade de pontos. O vetor gradiente calculado em cada ponto aponta para a região mais próxima da superfície. Deste modo, caso a superfície corte uma determinada aresta de uma célula, os gradientes em seus vértices (*voxels*) necessariamente devem apontar em sentidos opostos, enquanto em arestas onde não há interseção com a malha de triângulos, os gradientes dos seus vértices formam ângulos agudos pois apontam aproximadamente na mesma direção.

O cálculo do gradiente em um ponto $P(x, y, z)$ da grade de pontos pode ser feito através de dois métodos distintos, o método de diferenças finitas, ou de uma maneira mais simples, utilizando o algoritmo do cálculo da distância entre um ponto e um triângulo, descrito em: 5.2.

6.2.1 Diferenças Finitas

O gradiente da função de distância em um ponto $P(x, y, z)$ é dado por:

$$\nabla F(x,y,z) = \begin{bmatrix} \frac{|F(x+\Delta,y,z)| - |F(x-\Delta,y,z)|}{2\cdot\Delta} \\ \frac{|F(x,y+\Delta,z)| - |F(x,y-\Delta,z)|}{2\cdot\Delta} \\ \frac{|F(x,y,z+\Delta)| - |F(x,y,z-\Delta)|}{2\cdot\Delta} \end{bmatrix} \tag{4}$$

Onde Δ é o comprimento de uma aresta da célula. Note que neste cálculo, células vizinhas são visitadas para aquisição dos valores do campo de distância em seus pontos. Por isso, a equação 4 não pode ser aplicada em todos os pontos da grade: nos pontos de fronteira, algumas células vizinhas não existirão e então um diferente procedimento é empregado. Caso não haja célula vizinha, o gradiente do ponto é calculado a partir de *voxels* da mesma célula. As componentes x, y e z serão as seguintes:

$$F_x = \begin{cases} \dfrac{|F(x+\Delta,y,z)| - |F(x,y,z)|}{\Delta} & \text{se } x \text{ está no limite máximo da grade} \\ \dfrac{|F(x,y,z)| - |F(x-\Delta,y,z)|}{\Delta} & \text{se } x \text{ está no limite mínimo da grade} \end{cases} \tag{5}$$

$$F_y = \begin{cases} \dfrac{|F(x,y+\Delta,z)| - |F(x,y,z)|}{\Delta} & \text{se } y \text{ está no limite máximo da grade} \\ \dfrac{|F(x,y,z)| - |F(x,y-\Delta,z)|}{\Delta} & \text{se } y \text{ está no limite mínimo da grade} \end{cases} \tag{6}$$

$$F_z = \begin{cases} \dfrac{|F(x,y,z+\Delta)| - |F(x,y,z)|}{\Delta} & \text{se } z \text{ está no limite máximo da grade} \\ \dfrac{|F(x,y,z)| - |F(x,y,z-\Delta)|}{\Delta} & \text{se } z \text{ está no limite mínimo da grade} \end{cases} \tag{7}$$

6.2.2 Usando o Algoritmo de Distância Ponto Triângulo

Outra forma de calcular o gradiente do campo de distância é a de utilizar o algoritmo proposto em 5.2. O procedimento é bem simples, pois quando calculamos a menor distância entre um ponto e um triângulo, o algoritmo computa os valores s e t da equação paramétrica do triângulo: $T(s,t) = B + sE0 + tE1$. Com isso substituímos os valores de s e t nesta equação e obtemos o ponto pertencente ao triângulo que está mais próximo do ponto da grade em questão. Por fim, o vetor constituído pelos ponto da grade e o ponto do triângulo consiste no gradiente da função do campo de distância:

$$\nabla F(x,y,z) = P(x,y,z) - V(x,y,z) \tag{8}$$

Onde P é o *voxel* da grade e V o ponto do triângulo mais próximo de P.

Uma vez que os gradientes de todos os pontos da grade que representa o campo de distância foram calculados, para saber se a superfície corta uma dada aresta de uma célula, basta calcular o produto interno entre os gradientes dos dois

vértices desta aresta. Se o produto interno possuir um valor negativo, significa que os vetores apontam em sentidos opostos e com isso a superfície encontra-se entre eles. Caso o valor do produto interno seja positivo, os gradientes têm o mesmo sentido e portanto não há interseção entre a aresta e a malha de triângulos.

A visualização dos gradientes encontra-se na figura 9. O mesmo exemplo usado em 6.1 é considerado aqui. Desta vez, o método é capaz de diferenciar as células A e B quanto à existência de interseção da superfície ou não.

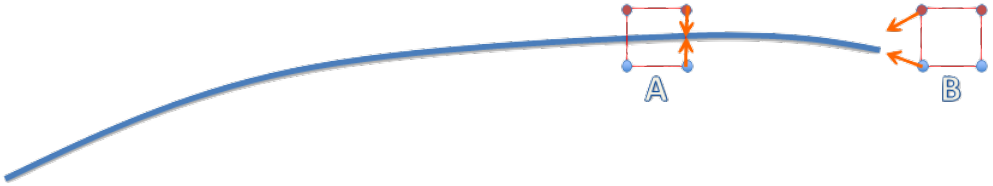

Figura 9: Os gradientes estão representados pelas setas em cor laranja. Desta vez é possível avaliar que as duas células se diferem. Enquanto na célula A os gradientes na aresta frontal direita apontam para sentidos opostos, na célula B os gradientes da mesma aresta apontam para o mesmo sentido.

6.3 Reposição dos Vértices

Finalmente, a última etapa deste algoritmo é a de atrair os vértices da malha recém gerada em direção à superfície original. Para realizar este procedimento, basta percorrer cada vértice da nova malha de triângulos e aplicar o seguinte algoritmo:

Algoritmo AtraiVertice( ):

for all vértice v da nova malha M' **do**

$dist \leftarrow DistanciaPontoMalha(v)$

$vetorGradiente \leftarrow GradienteCampoDistancia(v)$

$v.x = v.x - dist \cdot vetorGradiente.x$

$v.y = v.y - dist \cdot vetorGradiente.y$

$v.z = v.z - dist \cdot vetorGradiente.z$

end for

Com este algoritmo, os vértices da nova malha de triângulos são deslocados no sentido da superfície original, obtido através do cálculo do gradiente do campo de distância nas coordenadas do vértice v. A distância cujo o vértice em questão irá se deslocar é determinada pela multiplicação do fator $dist$.

6.3.1 Imperfeições no Algoritmo

Este algoritmo de reconstrução de malha de triângulos proposto por [Perry e Frisken 2001] considera apenas aplicações à superfícies fechadas, como por exemplo: renderizações volumétricas de animais e peças mecânicas. Entretanto,

no escopo deste projeto, o algoritmo irá atuar em superfícies geológicas, que constituem superfícies abertas.

As imperfeições criadas na superfície recriada neste algoritmo ocorre nas bordas. A figura 10 ilustra a seguinte situação: a superfície, de cor cinza e delimitada pela fronteira em azul, é vista de cima. Duas células, A e B, são consideradas para a reconstrução da malha de triângulos e possuem pontos centrais identificados pelos círculos em vermelho. O procedimento que será realizado neste momento registrado pela figura 10 é o da atração dos vértices dos triângulos da nova malha gerada em direção à superfície original. Note que no caso da célula B o vértice será atraído corretamente, pois o seu gradiente, representado pela seta em azul, aponta no sentido correto de onde o limite da superfície se encontra. Porém, no caso da célula A, o mesmo cenário não ocorre: por estar acima da superfície original, o gradiente do vértice central desta célula aponta na direção perpendicular à esta superfície, descrito pelo vetor em vermelho (exibido como um "X" circunscrito), ao invés de apontar para a borda da superfície, tal qual o vetor em amarelo sugere.

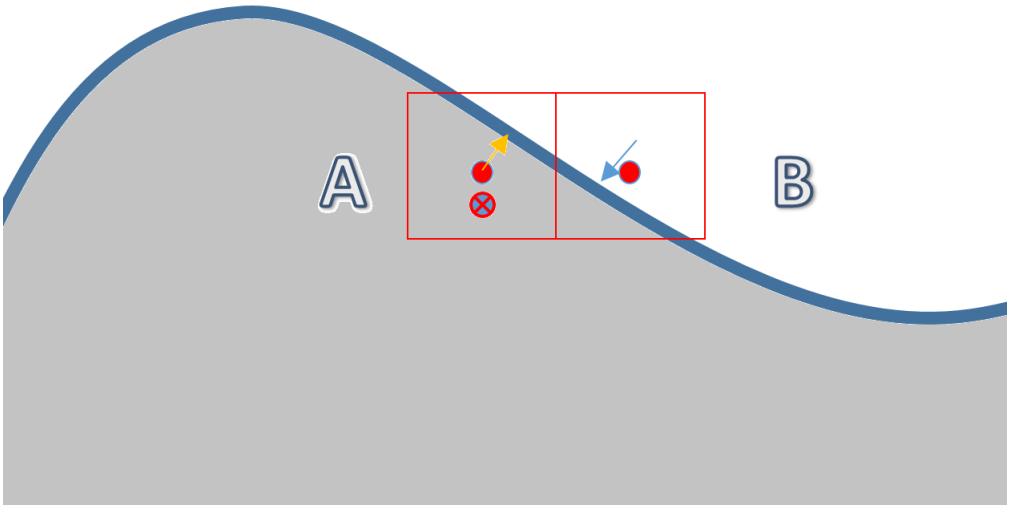

Figura 10: Atração incorreta dos vértices da nova malha gerada. Enquanto o vértice da célula B é devidamente atraído para a borda da superfície original, o mesmo não ocorre com o vértice da célula A, que é movido perpendicularmente para a superfície ao invés de se locomover em direção à borda.

Esta atração incorreta resulta em bordas serrilhadas (*aliasing*), uma vez que nem todos os vértices dos triângulos da nova superfície que representam a fronteira da malha de triângulos original se localizam nas coordenadas corretas. A figura 11 exemplifica esta situação. O triângulo em cor vermelha é o triângulo formado após a atração incorreta descrita no parágrafo anterior, enquanto o triângulo em amarelo consiste no triângulo pertinente a ser criado, visto que seus vértices correspondentes às células localizadas na fronteira da superfície foram atraídos

para a borda, não existindo neste caso *aliasing* algum.

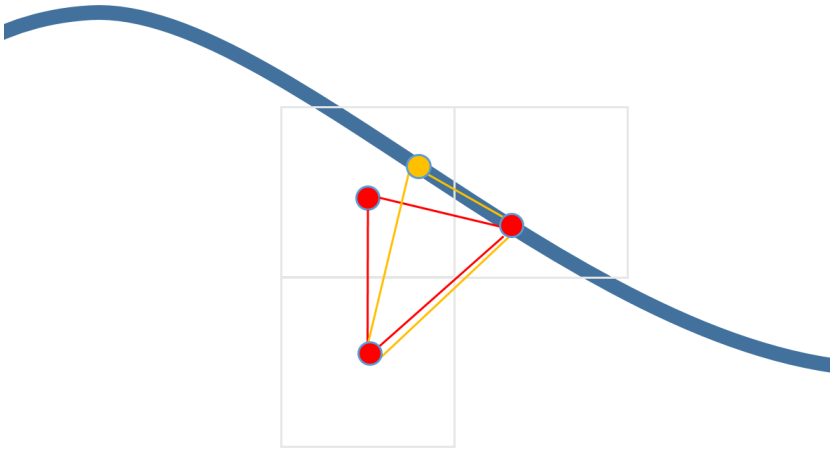

Figura 11: Criação imperfeita dos triângulos ao longo da fronteira da superfície, gerando bordas serrilhadas.

A seguir encontram-se dois exemplos de superfícies regeradas, representadas pelas figuras 12, 13, 14 e 16 que ilustram o resultado gerado pelo algoritmo de reconstrução de malha e sua imperfeição. A primeira imagem consiste na malha de triângulos original, enquanto a imagem seguinte representa a superfície reconstruída, notando-se suas bordas contendo *aliasing*.

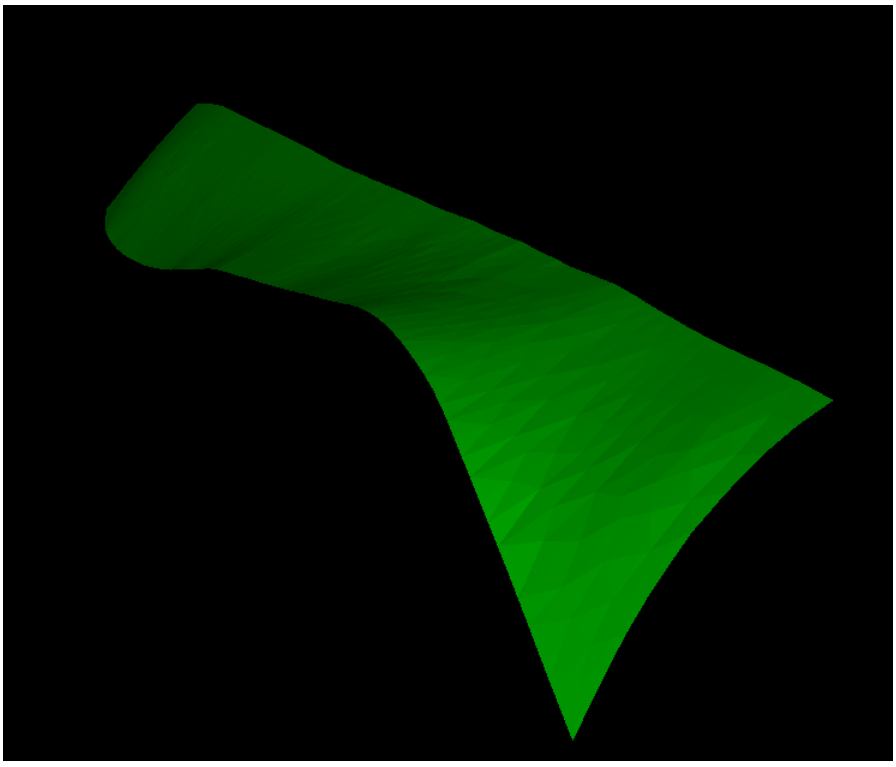

Figura 12: Superfície “Extensão” original.

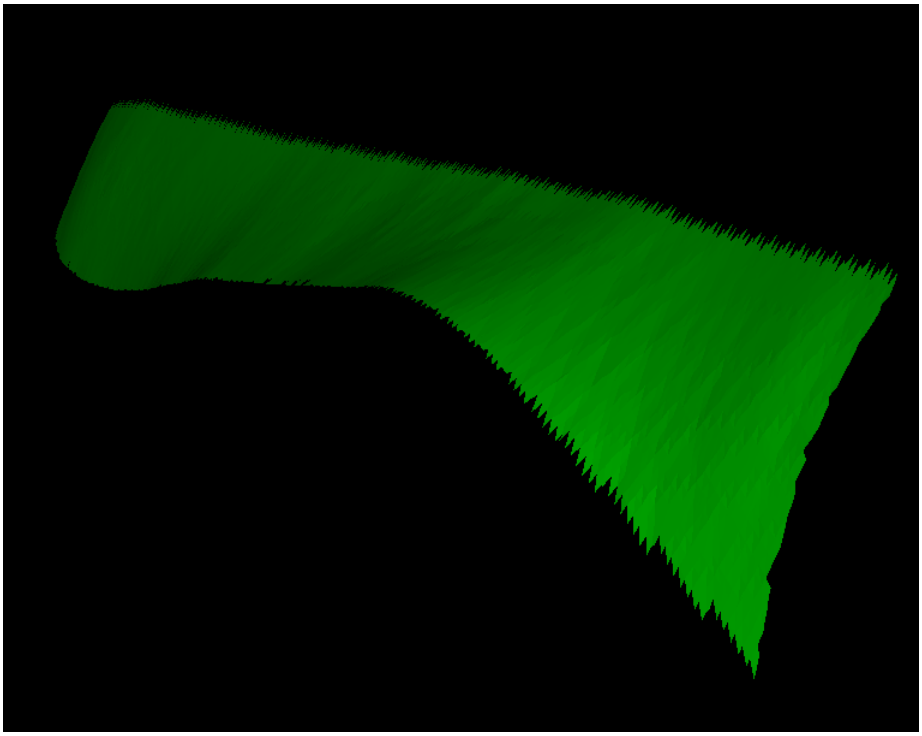

Figura 13: Superfície “Extensão” recriada pelo algoritmo.

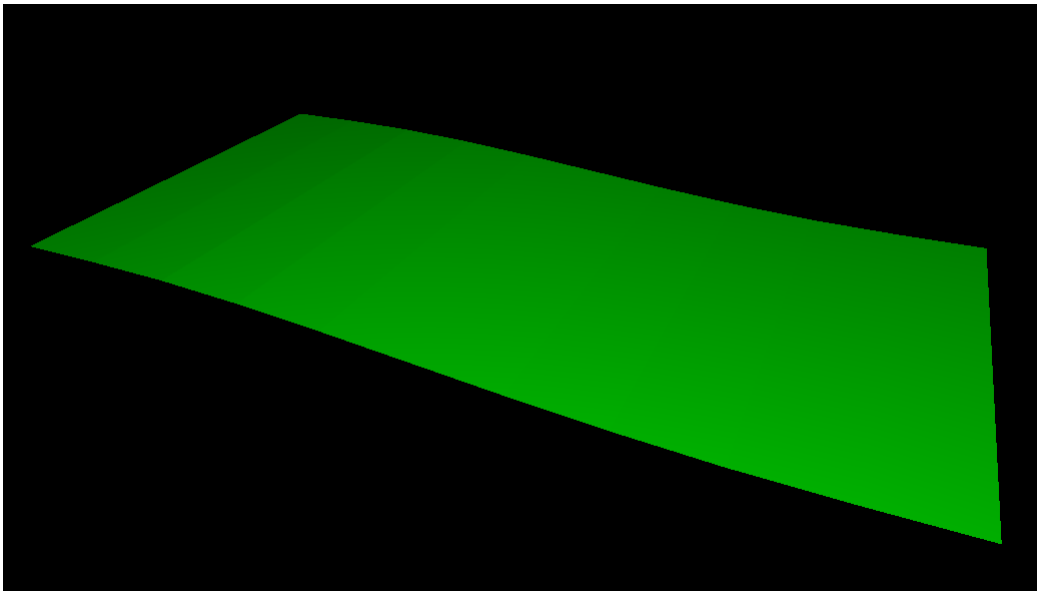

Figura 14: Superfície “ss_44” original.

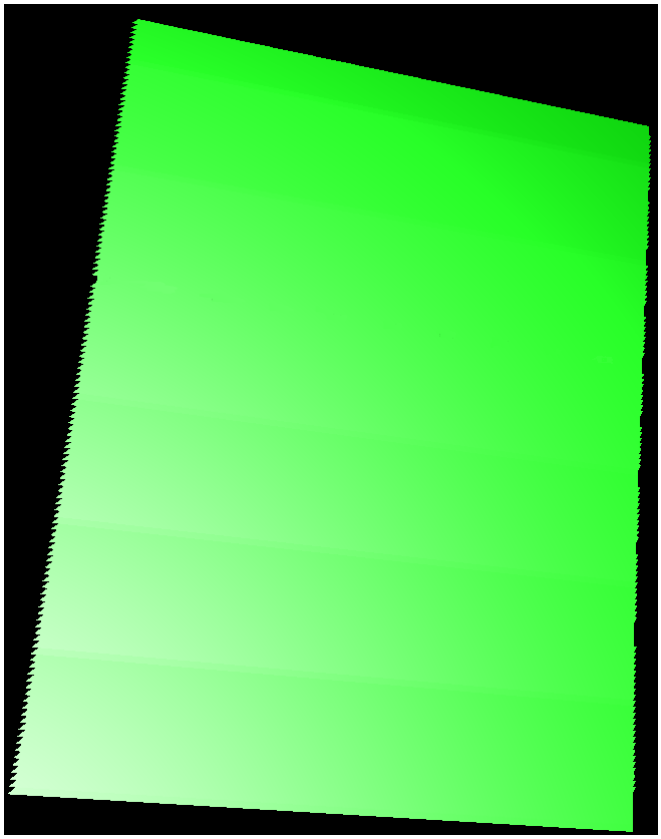

Figura 15: Superfície “ss_44” recriada pelo algoritmo.

Figura 16: Detalhe da borda serrilhada da superfície “ss_44” recriada.

6.3.2 Correção do *Aliasing*

Para evitar o serrilhamento apresentado anteriormente, propõe-se aqui a seguinte solução:

As imprecisões do algoritmo ocorrem nas bordas da superfície. De fato, é possível descobrir se uma célula do campo de distância está mais próxima de um vértice de borda da superfície original. No algoritmo sugerido em 5.2, além de realizar o cálculo de distância entre um ponto da grade até a superfície original, também é explicitado se o ponto do triângulo mais próximo do *voxel* encontra-se dentro do triângulo, em um vértice ou em uma aresta do triângulo. Ou seja, se listarmos de antemão todos os vértices da malha de triângulos original que se localizam na borda da mesma, saberemos exatamente quando atuar de maneira diferenciada na atração dos vértices da malha regerada. Assim, ao executarmos o algoritmo de 5.2, podemos saber se o ponto do triângulo mais próximo do ponto da grade consiste em um vértice ou está em uma das arestas do triângulo. Se for o primeiro caso, basta verificar se este vértice pertence à lista dos vértices de borda. Se for o segundo, deve-se averiguar se os dois vértices nas extremidades desta aresta estão na borda da superfície. Desta maneira, conseguimos identificar se um dado *voxel* da grade possui como ponto mais próximo da malha de triângulos um vértice em sua borda.

Na estrutura CSI, cada vértice de uma superfície guarda uma lista de triângulos cujo o vértice em questão faz parte. A partir desta propriedade da estrutura, o pseudocódigo do algoritmo que identifica vértices de borda realiza o seguinte procedimento:

Algoritmo éVerticeBorda(Vertice x):

if número triângulos de x = 1 **then**
  return TRUE
end if
for all Triângulo T de x **do**
  for all Vértice v de $T \neq x$ **do**
    $count \leftarrow 0$
    for all Triângulo T' de v **do**
      if T' contém x **then**
        $count++$
      end if
    end for
    if $count = 1$ **then**
      return TRUE
    end if
  end for
end for
return FALSE

Com isso, iremos marcar todos os pontos da grade para identificarmos quais estão mais próximos da borda da superfície.

Após passar pelo processo de criação dos triângulos da superfície reconstruída, o algoritmo de atração dos vértices desta superfície em direção à malha original deve ser um pouco diferente. Em vez de tomar somente um gradiente aproximado no centro da célula como referência para a direção da atração do vértice, faremos uma média de todos os gradientes da célula que apontam para a borda da superfície original, e utilizamos o gradiente resultante desta média como a orientação a ser seguida para a atração do vértice.

Caso nenhum ponto da célula seja mais próxima de uma borda da malha de triângulos original, fazemos primeiramente a atração individual de cada ponto da célula, utilizando seus respectivos gradientes e valores da função do campo de distância. Em seguida, obtemos a média das novas coordenadas destes oito pontos e substituímos as coordenadas do vértice central da célula com esta nova coordenada calculada.

Assim, as bordas da superfície recriada se tornam mais suaves, como podemos ver no resultado da figura 18.

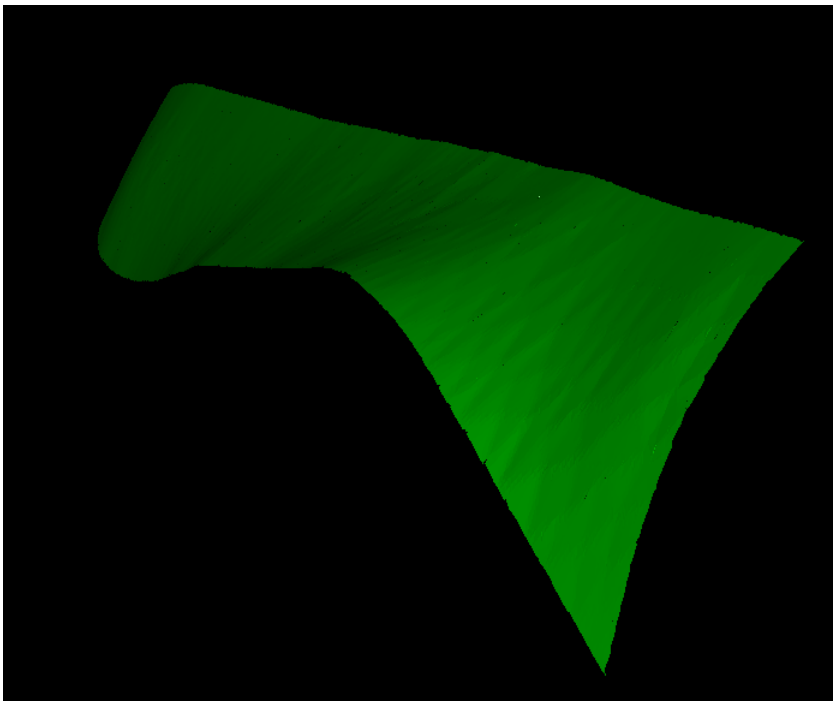

Figura 17: Superfície “Extensão” recriada, com tratamento contra o serrilhado.

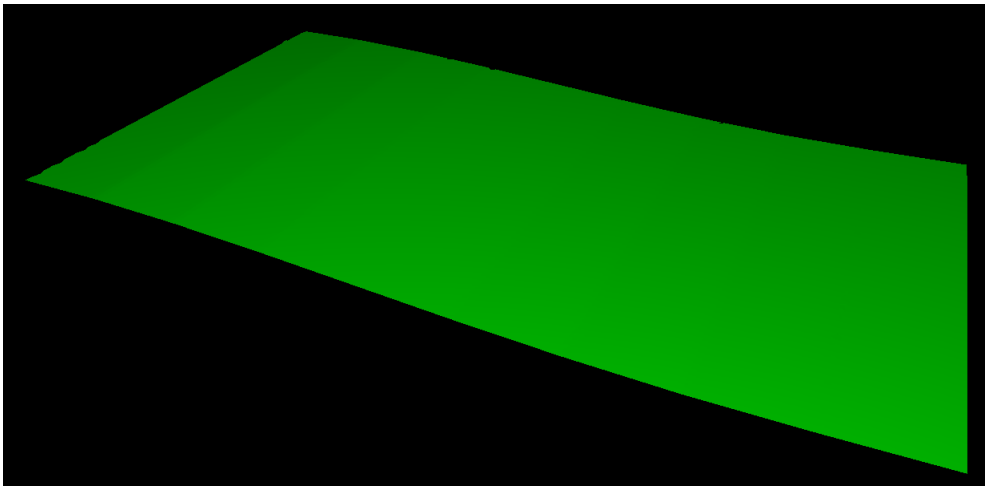

Figura 18: Superfície “ss_44” recriada, com tratamento contra o serrilhado.

6.4 Persistência dos Dados

Como descrito em 2.2, a complexidade do algoritmo de força bruta para calcular o campo de distância é problemática, pois se a malha em questão possuir um grande número de triângulos, o tempo para realizar esta rotina se torna longo. Logo, uma funcionalidade importante que a biblioteca ReMGeo oferece é a de armazenar todas as informações do campo de distância em um arquivo de saída, logo após o cálculo deste campo ser computado. Desta maneira, a aplicação que fornece a visualização da superfície e seu campo de distância não necessita desempenhar o cálculo do campo de distância toda vez que ela é executada. O módulo de persistência dos dados irá carregar o arquivo contendo as informações do campo de distância e construir a grade contendo todos os *voxels* e seus valores da função de distância.

O formato do arquivo é bem simples. A primeira informação contida no arquivo é a do valor do passo (*size*), essencial para que os pontos da grade criados estejam espaçados corretamente. Em seguida encontram-se os valores das menores distâncias de cada *voxel* da grade até a superfície. Após todos estes dados serem processados, a aplicação já está pronta para exibir na tela a malha de triângulos e seu campo de distância, sem haver o desperdício de recalcular esta informação.

Abaixo encontra-se um exemplo do arquivo gerado pelo módulo de persistência de dados da biblioteca ReMGeo. A extensão do arquivo é padronizada: “.df”.

```
size = 80
# BEGIN VOXELS
3192.45
3146.72
3101.06
```

3055.76
3010.57
2965.57
2920.78
2876.12
2831.79
2787.59
2743.19
−2854
−2900.66
−2947.34
−2994.19
−3041.13
−3088
−3134.76
−3181.33
−3227.69
END VOXELS

7 Implementação e Avaliação

7.1 Planejamento e Execução de Testes Funcionais

7.1.1 Teste do Cálculo de Distância

Para garantir a total corretude do código descrito em 5.2, tornou-se necessário criar uma unidade de teste. A seguir encontram-se informações deste módulo de teste.

Basicamente, um triângulo pré-determinado é utilizado para uma intensa bateria de testes. Diferentes localidades do espaço são consideradas para assegurar que todas as sete regiões descritas na figura 2 sejam devidamente testadas.

Para que a complexidade do teste não seja substancial, apenas os pontos cujas distâncias até o triângulo já são conhecidas a priori são considerados. Assim que o cálculo da distância é concluído pelo algoritmo desenvolvido neste projeto, o valor da distância encontrado pode ser prontamente comparado com o valor real da distância entre o ponto dado e o triângulo. Qualquer distância que porventura seja calculada incorretamente pelo algoritmo será imediatamente reportada pela unidade de teste, que indica onde foi o erro e qual a discrepância entre o valor correto e o valor encontrado da distância.

É importante ressaltar que por pequenas imprecisões computacionais em alguns casos, uma tolerância (que pode ser definida pelo usuário) é necessária no momento da comparação dos valores encontrados pelo algoritmo e os valores

reais da distância.

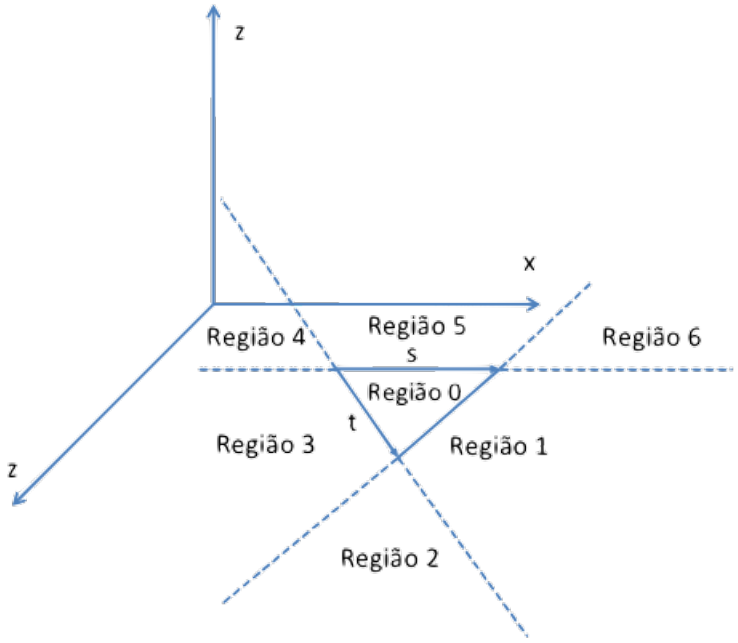

Figura 19: Ilustração do ambiente de teste do cálculo de distância de um ponto ao triângulo.

A figura 19 auxilia na demonstração de como o teste é feito. Primeiramente, partindo de um vértice do triângulo, o teste seleciona pontos que estão ao longo do eixo z para fazer os cálculos, além dos pontos contidos nas fronteiras (representados pelas linhas tracejadas na figura) que representam importantes casos de teste. Em seguida, pontos que se encontram ao longo da bissetriz de cada par de linhas tracejadas também são levados em consideração para teste. Isto é feito para todos os três vértices do triângulo. Por fim, para testar as arestas, calcula-se a distância dos pontos que estão contidos nas retas que são perpendiculares à cada uma das três arestas até o triângulo.

7.2 Paralelização do Cálculo de Distância

Computação paralela é um método de programação em que se torna possível realizar procedimentos de maneira simultânea [A.Gottlieb e Almasi 1989], partindo do princípio de que problemas grandes podem ser divididos em outros menores, que por sua vez são solucionados concorrentemente (em paralelo).

Esta técnica tem sido empregada principalmente na área de computação de alta performance, mas com o advento da limitação do dimensionamento da frequência de *clock* dos processadores, a computação paralela vem ganhando cada vez mais atenção no mundo da programação ["Parallel Computing Research at Illinois: The UPCRC Agenda"]. Por causa disto, os processadores de múltiplos núcleos conquistaram bastante espaço no mercado, pois além de oferecerem

grande performance computacional, também contribuem para um menor consumo de energia e consequentemente menor dissipação de calor.

Entretanto, um programa de computador que utiliza métodos de paralelização possui uma desvantagem: escrevê-lo é mais complicado do que criar algoritmos seriais. Em geral, é mais difícil escrever códigos que realizam procedimentos em paralelo, pois precisamos levar em consideração como cada *thread* (linha ou encadeamento de execução) irá operar no algoritmo em questão, assegurando assim que não haja conflito na utilização de uma parte da memória por *threads* diferentes. De qualquer maneira, uma vez que a implementação desta estratégia é finalizada e testada, percebe-se uma diminuição considerável no tempo de execução do cálculo.

Neste projeto, a rotina que calcula a distância de cada ponto da grade à superfície, descrito em 6.3.2, conta com duas diferentes implementações de paralelização, o OpenMP e Pthreads. O OpenMP é uma API que suporta programação paralela em diversas linguagens, como: C, C++ e Fortran, em várias plataformas. É uma API bem simples que consiste em diretivas de compilador, rotinas de bibliotecas e variáveis de ambiente que influenciam no comportamento de execução. O OpenMP é uma API de alto nível e por isso sua utilização é em muitas das vezes trivial. Por outro lado, o Phtreads é uma API de baixo nível e por conseguinte sua implementação costuma ser bem mais desafiadora. Entretanto, Pthreads oferece um controle extremamente fino no gerenciamento das *threads* (criação e junção), exclusões múltiplas e outros.

7.3 Arquivos de Superfícies Triangulares para Teste e Resultados

A reconstrução de malhas triangulares foi testada em modelos GOCAD existentes. Todos estes modelos são provenientes do laboratório Tecgraf PUC-Rio, criados justamente com o intuito de testar outras aplicações desenvolvidas pelo próprio laboratório. A tabela 2 lista as superfícies testadas e suas propriedades, além da dimensão do espaçamento entre *voxels* da grade e os tempos dos cálculos do campo de distância utilizando o algoritmo sequencial e os algoritmos paralelos. Foram utilizadas 8 *threads* na execução dos testes usando OpenMP e Pthreads.

Superfície	No. Triângulos	No. Vértices	size	Tempo Cálculo Sequencial (seg)	Tempo Cálculo Paralelo OpenMP (seg)	Tempo Cálculo Paralelo Pthreads (seg)	Ganho OpenMP / Pthreads
Extensão	944	565	20	69.94	32.79	37.01	53.12% / 47.08%
Sal topo	3752	1965	80	139.75	64.44	67.42	53.89% / 51.76%
Tampa1	144	94	40	29.35	12.37	12.66	57.85% / 56.87%

Tabela 2: Superfícies Testadas e Resultados

7.4 Aplicação: Visualizador de Resultados

Este projeto compreende um visualizador gráfico tridimensional que provê as seguintes funcionalidades visuais: a superfície original, o campo de distância (com

ou sem sinal),os vetores gradientes do campo de distância, os vértices localizados no centro de cada célula da grade e a superfície regerada.

Para executar a aplicação, é necessário passar como argumento qual superfície deseja ser carregada. Se o arquivo passado para o programa possuir extensão ".ts", significa que o usuário deseja carregar a superfície original e computar seu campo de distância. Logo, ele deverá executar, por exemplo:

```
$ ../bin/MacOS109/appd surface.ts
```

Se o usuário já possuir o arquivo ".df" de sua superfície original, ele pode executar a aplicação passando este arquivo para que assim não seja necessário calcular o campo de distância novamente, visto que o arquivo ".df" já contem toda a informação do campo de distância, como especificado em 6.4. As superfícies reconstruídas são automaticamente salvas em um arquivo à parte, com o nome do arquivo original acrescido do sufixo "_NET". Se desejar, o usuário também pode carregar este arquivo diretamente, resultando na visualização apenas da malha recriada.

7.4.1 Atalhos de Teclado

Para habilitar ou desabilitar as funcionalidades enumeradas acima, a aplicação conta com atalhos de teclado. Abaixo está uma lista dos atalhos:

- Habilitar/desabilitar visualização da superfície original: tecla "m".
- Visualizar *voxels* do campo de distância: tecla "d".
- Alternar a visualização do campo de distância (*voxels*) entre sem sinal e com sinal: tecla "s"
- Visualizar vetores gradientes do campo de distância: tecla "g".
- Visualizar vértices localizados no centro de cada célula: tecla "c".
- Visualizar superfície regerada: tecla "w".
- Aumentar/diminuir *zoom*: teclas "+" e "-".
- Visualizar apenas triângulos de borda da superfície: tecla "b".

Para exemplificar as visualizações citadas acima, seguem as figuras abaixo:

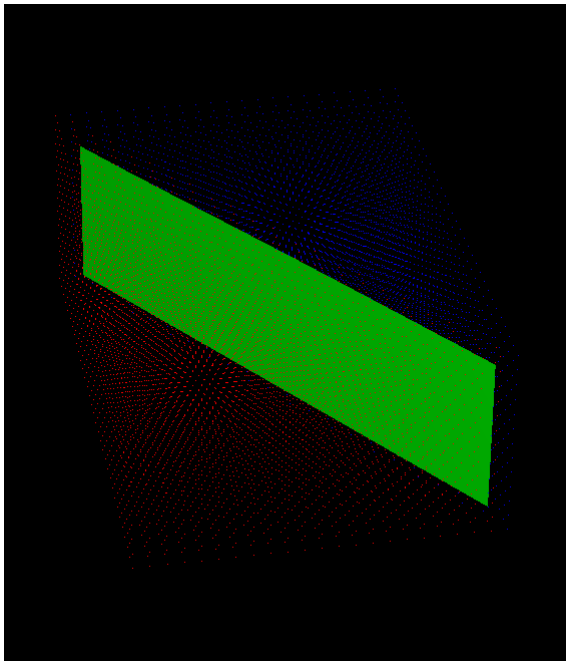

Figura 20: Visualização do campo de distâncias da superfície Lado1. Os pontos do campo que estão com sinal positivo foram desenhados com cor vermelha, e os com sinal negativo na cor azul.

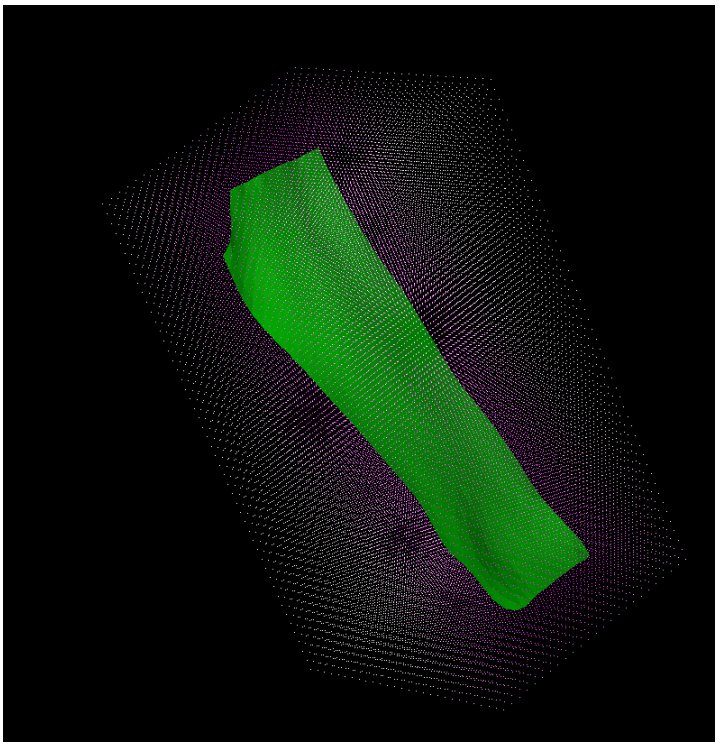

Figura 21: Visualização do campo de distâncias sem considerar o sinal, na superfície Graben de Marlin. Os pontos na cor magenta representam os pontos da grade com menor valor em módulo do campo de distância.

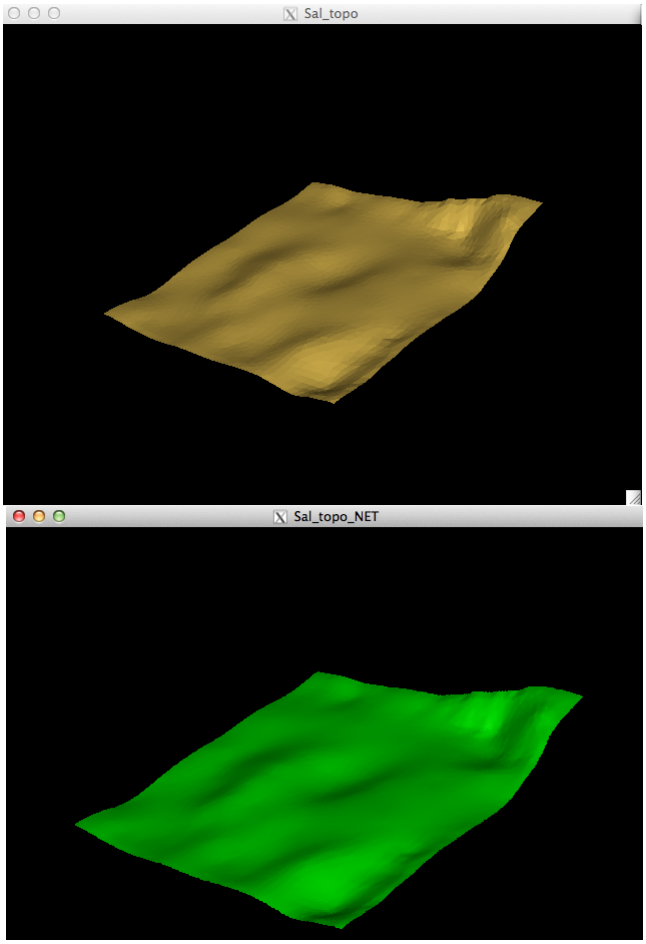

Figura 22: A superfície Topo Sal original em cor dourada, e a mesma superfície reconstruída em cor verde, utilizando o algoritmo descrito em 6.

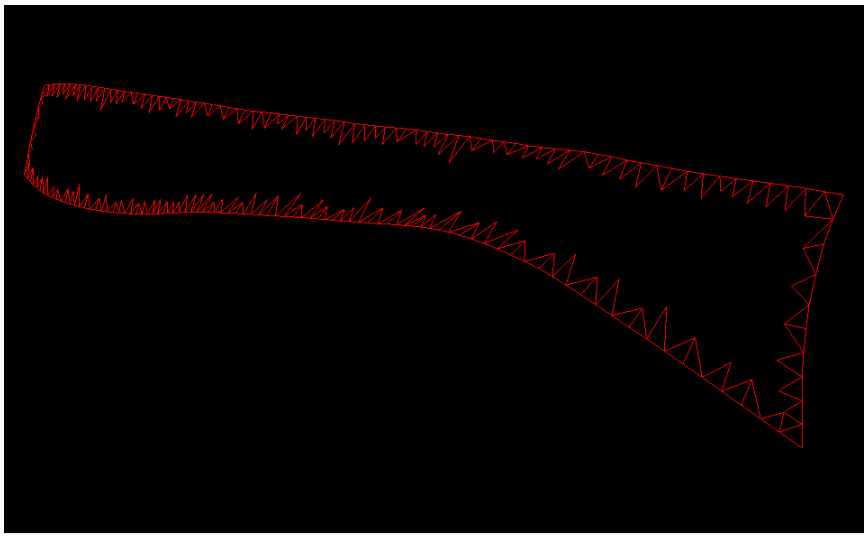

Figura 23: Visualização dos triângulos de borda da superfície Extensão.

8 Considerações Finais

Este projeto conseguiu demonstrar que campos de distância são capazes de representar superfícies de maneira consistente, sem utilizar muitos recursos de memória e de tempo. Apesar de o algoritmo de força bruta não resultar em altas performances, ele conta com a maior precisão possível. De qualquer forma, este trabalho contribuiu como um passo inicial para a integração de campos de distâncias com problemas relacionados à Geologia Estrutural, para que geólogos e engenheiros recebam melhor assistência na interpretação e manipulação de modelos geológicos.

8.1 Trabalhos Futuros

O alto custo computacional do algoritmo de força bruta acarretou na procura de métodos de aproximação, sendo o algoritmo *Chamfer Distance Transform* uma das alternativas. Uma possível implementação é a de dividir num processo de dois passos. O passo para a frente calcula as distâncias a partir da superfície indo em direção à parte inferior do conjunto de dados, enquanto o passo para trás calcula as distâncias remanescentes.

Esta aproximação ao problema possui baixo custo porque cada *voxel* só é considerado duas vezes, e seu cálculo depende da adição de elementos da matriz de distância com seus vizinhos. Porém, é um método relativamente impreciso e por isso é apenas recomendado para aplicações onde precisão não é prioridade.

Se a precisão for realmente crucial no cálculo do campo de distância de uma determinada superfície, podemos manter o algoritmo de força bruta mas utilizar outro método bastante diferente de programação. O recurso é chamado de GPGPU (sigla em inglês para *General-purpose computing on graphics processing units*), que faz uso da placa gráfica do computador. Placas de vídeo manipulam computações relacionadas à área de computação gráfica, porém a ideia aqui é aproveitar o alto poder computacional deste dispositivo em aplicações normalmente realizadas pela CPU. [*Proceedings of the International Symposium on Wearable Computing* 2002] [*Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing* 2004]

Placas gráficas têm a capacidade de processar fragmentos de códigos independentes, sendo que muitos deles em paralelo. Por causa disso, elas são chamadas de *stream processors* (processadores de fluxo contínuo). Porém, é importante que a aplicação em questão possua alta intensidade aritmética e independência dos dados, caso contrário ocorrerá uma alta latência no acesso à memória e consequentemente haverá uma limitação considerável no ganho de performance [Asanovic e Bodik 2011]. Felizmente, no caso deste projeto, o cálculo do campo de distância consiste em computar a distância individual de cada ponto da grade até a superfície, o que significa que não há alguma dependência entre o cálculo de

diferentes *voxels* da grade. Logo, é bem provável que aplicar técnicas de GPGPU na biblioteca ReMGeo trará resultados positivos, diminuindo o tempo necessário para calcular o campo de distância.

Referências

Adve, S.V. e J.C. Hart. "Parallel Computing Research at Illinois: The UPCRC Agenda". Em: *Parallel@Illinois*.

A.Gottlieb e G.S. Almasi (1989). *Highly Parallel Computing*. Benjamin/Cummings.

Asanovic, K. e R. Bodik (2011). "A view of the parallel computing landscape". Em: *Communications of the ACM* 52.10, pp. 56–67.

Eberly, D. (2008). *Distance Between Point and Triangle in 3D*. URL: http://www.geometrictools.com/Documentation/DistancePoint3Triangle3.pdf.

Hoffmann, C.M. (1989). *Geometric and Solid Modeling*. Morgan Kaufmann.

Jones, M. W., J. A. Bæsrentzen e M. Sramek (2006). "3D Distance Fields: A Survey of Techniques and Applications". Em: *IEEE Transactions on Visualization and Computer Graphics* 12.4, pp. 581–599.

Jones, M. W. e R. Satherley (2001). "Using Distance Fields for Object Representation and Rendering". Em: *Eurographics*.

Paradigm. *Gocad*. URL: http://www.pdgm.com/Products/GOCAD.

Perry, R. N. e S. F. Frisken (2001). *Kizamu: A System For Sculpting Digital Characters*. Rel. téc. Mitsubishi Electric Research Laboratories.

Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (2004).

Proceedings of the International Symposium on Wearable Computing (2002).

PUC-Rio, Tecgraf. *Modelagem Geológica de Sistemas Petrolíferos*. URL: http://www.tecgraf.puc-rio.br/areas-de-pesquisa/modelagem-geologica-de-sistemas-petroliferos/.

— *Tecmake Build Tool*. URL: http://www.tecgraf.puc-rio.br/tecmake/.

Rosenfeld, A. e J.L. Pfaltz (1966). "Sequential Operations in Digital Picture Processing". Em: *J. ACM* 13.4, pp. 471–494.

Vanco, M. e H.O. Kreylos (2011). "Distance Field Computation for Geological Slab Surface Data Sets". Em: *Computing and Visualization in Science* 14.4, pp. 143–156.