Ricardo Cavalcanti Marques

Algoritmo Robusto para Interseção de Superfícies Triangulares

Dissertação de Mestrado

Dissertação de Mestrado apresentada ao Curso de Mestrado em Informática como requisito parcial para a obtenção do titulo de Mestre em Informática.

Orientador: Prof. Hélio Côrtes Vieira Lopes

Coorientador: Prof. Luiz Fernando Campos Ramos Martha

Rio de Janeiro

Setembro de 2013.

Ricardo Cavalcanti Marques

Algoritmo Robusto para Interseção de Superfícies Triangulares

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre pelo Programa de Pós-graduação em Informática do Departamento de Informática do Centro Técnico e Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Hélio Côrtes Vieira Lopes
Orientador
Departamento de Informática – PUC-Rio

Prof. Luiz Fernando Campos Ramos Martha
Coorientador
Departamento de Eng. Civil – PUC-Rio

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

Márcio Rodrigues de Santi
Instituto TeCGraf

Prof. José Eugenio Leal
Coordenador Setorial do Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, 06 de Setembro de 2013

Ricardo Cavalcanti Marques

Graduou-se em Engenharia de Computação na PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro) no final de 2010. Em 2007, realizou pesquisa de iniciação científica na área de Análise Numérica. Desde 2009, vem desenvolvendo, no Laboratório Tecgraf (atualmente Instituto Tecgraf), uma vasta gama de ferramentas de modelagem para o *software* Mtool3D. Participou de dois congressos internacionais na área de Mecânica Computacional. Sua linha de pesquisa atual consiste em introduzir a tecnologia de aritmética exata a algoritmos clássicos da Geometria Computacional.

Marques, Ricardo Cavalcanti

Algoritmo robusto para interseção de superfícies triangulares / Ricardo Cavalcanti Marques ; orientador: Hélio Côrtes Vieira Lopes ; coorientador: Luiz Fernando Campos Ramos Martha. – 2013.

117 f. : il. (color.) ; 30 cm

Dissertação (mestrado)–Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2013.

Inclui bibliografia

1. Informática – Teses. 2. Interseção de surpefícies. 3. Algoritmos robustos. 4. Aritmética exata. I. Lopes, Hélio Côrtes Vieira. II. Martha, Luiz Fernando Campos Ramos. III. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

Ao eterno “vovôzão”, Fernando Augusto Marques
(in memoriam), pela força e alegria contagiantes

Agradecimentos

Primeiramente, ao meu orientador, Hélio Lopes, pela sua paciência e incentivo que tornaram possível a conclusão desta dissertação. Nossos encontros semanais, dosados por sua vasta experiência e sabedoria, foram fundamentais para me guiar nesta jornada, mesmo quando tudo parecia dar errado.

Ao meu coorientador, Luiz Fernando Martha, por sempre acreditar em meu potencial e me propor o desafio de desenvolver ferramentas robustas de modelagem geométrica, em particular a Interseção de Superfícies, tema deste trabalho.

Ao meu antigo colega do Tecgraf, André Brabo, que me iniciou na área de modelagem geométrica e me apresentou à Aritmética Exata.

Aos membros da banca, Waldemar Celes e Luiz Henrique Figueredo, pelos suas valorosas críticas na defesa de proposta, que se revelaram essenciais a este trabalho.

Ao Tecgraf, pelo auxílio financeiro, sem o qual esta dissertação jamais seria possível.

Aos meus pais, cujo interesse em minha dissertação me foi fonte de motivação.

A todos os meus colegas do Tecgraf, por providenciar um clima agradável e colaborativo, propício para o desenvolvimento desta dissertação.

A toda a família e amigos, pelo incentivo e apoio constantes. Em especial, aos meus amigos Carlos e Marcell pelo seu grande interesse.

A todos os professores e colegas da PUC-Rio, que foram tão importantes na minha vida acadêmica.

Resumo

Marques, Ricardo Cavalcanti, Lopes, Hélio Côrtes Vieira, Martha, Luiz Fernando Campos Ramos **Algoritmo Robusto para Interseção de Superfícies Triangulares**. *Rio de Janeiro, 2013, 117p. Dissertação de Mestrado – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.*

O objetivo deste trabalho é projetar e implementar um algoritmo eficiente, confiável e preciso para a interseção de superfícies triangulares que representam modelos geológicos complexos. A grandeza das coordenadas espaciais desses modelos, em contraste com o relativamente pequeno tamanho médio de seus elementos, levam a problemas numéricos que podem gerar modelos ruins ou a erros graves do modelador geométrico. Além disso, um alto nível de precisão é desejável para se evitar erros de modelagem que possam gerar acidentes no campo de exploração. Neste trabalho, é proposta uma solução para reduzir os problemas numéricos com o uso de algumas estratégias geométricas e da Aritmética Exata. Exemplos demonstram estes problemas de robustez e validam o algoritmo proposto.

Palavras-Chave

Interseção de superfícies; algoritmos robustos; aritmética exata

Abstract

Marques, Ricardo Cavalcanti, Lopes, Hélio Côrtes Vieira (Advisor), Martha, Luiz Fernando Campos Ramos **Robust Algorithm for Triangulated Surfaces Intersection**. *Rio de Janeiro, 2013, 117p. M.Sc. Thesis – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.*

The goal of this work is to design and to develop an efficient, reliable, and accurate algorithm for the intersection of triangular surfaces that represent complex geological models. The wide range of these models' coordinates in contrast with the relatively small average size of its elements lead up to numerical instability problems, which may generate bad models or crash the geometric modeler. Additionally, a high degree of precision is desired in the model to avoid accidents in the field of oil exploration. In this work, it is proposed a solution to reduce the numerical issues by the use of some geometrical strategies and the Exact Arithmetic. Examples are used to demonstrate these robustness problems and to validate the proposed algorithm.

Keywords

Surface-to-surface intersection; robust algorithms; exact arithmetic

Sumário

Lista de Figuras

Lista de Tabelas

O sucesso nasce do querer, da determinação
e persistência em se chegar a um objetivo.
Mesmo não atingindo o alvo, quem busca
e vence obstáculos, no mínimo fará coisas
admiráveis.
José de Alencar

1

Introdução

1.1. Motivação

A solução de problemas de Engenharia passa, em muitas vezes, pela construção de modelos virtuais para análise e simulação. Dessa forma, é possível realizar um estudo detalhado acerca do problema em seu estado atual e até prever as possíveis consequências decorrentes das ações tomadas. Mais do que simplesmente fornecer apoio a uma decisão instantânea, isto possibilita o planejamento em longo prazo e o contínuo controle do cumprimento do plano estabelecido. Se forem detectadas eventuais divergências em relação ao plano traçado, será possível ainda minimizá-las, senão corrigi-las totalmente, implantando ações baseadas em novos estudos e simulações.

Em especial na indústria do petróleo, esse planejamento é essencial para viabilizar a exploração e produção, definindo estratégias de perfuração, prospecção, e contenção sem que haja acidentes ambientais, danos materiais e/ou perdas humanas. Portanto, é fundamental prover meios para a criação de modelos que expressem bem os elementos encontrados no mundo real.

Todavia, a qualidade dos modelos gerados depende não somente da perícia do usuário, mas também da existência de ferramentas robustas de modelagem no aplicativo em questão. De fato, tais ferramentas facilitam o processo de modelagem, uma vez que possibilitam técnicas mais simples (e nem, por isso, menos eficazes) de se modelar.

Para providenciar soluções rápidas na ausência de tais ferramentas, alguns problemas são estudados com modelos simplificados, os quais, infelizmente, podem perder muito realismo. A exclusão de entidades do modelo deve ser feita com muito cuidado, para não banir entidades com atuações importantes. Em alguns casos, é difícil prever, antes da simulação, quais entidades terão papéis mais relevantes.

É importante ressaltar que essas ferramentas podem automatizar tarefas repetitivas, geralmente desgastantes ao usuário. Nesse ponto, convém relembrar que quanto menos árduo for o trabalho do usuário, menores serão as chances de ocorrer falhas humanas, evitando remodelagens. Assim, a eficiência e a confiabilidade do processo como um todo podem ser vertiginosamente aumentadas.

O modelador geométrico, então, deve disponibilizar ferramentas robustas e intuitivas para que o usuário se sinta confortável a utilizá-lo. O nome e utilização dessas ferramentas também devem ser bem óbvios, para não deixar o usuário com dúvidas ou dificuldades ou, pior, levá-lo ao manuseio incorreto da ferramenta.

Ainda que o usuário faça um uso adequado dessas ferramentas, se elas não funcionarem corretamente, a modelagem ficará comprometida. Daí, surge a obrigatoriedade de robustez nessas ferramentas, que será um dos focos deste trabalho. Em última análise, a existência de boas ferramentas, então, torna possível a modelagem, em tempo hábil e com segurança, de cenários cada vez mais complexos e desafiadores.

1.2. Objetivos

Nesse sentido, este trabalho se propõe a projetar e implementar um algoritmo robusto, inspirado em LO & WANG (2005), para a interseção de superfícies que representam estruturas geológicas. Dado o grande domínio tridimensional que os modelos geológicos encobrem e, por vezes, seu elevado grau de complexidade, surgem desafios tais que os algoritmos tradicionais com este fim merecem ser revistos. Entende-se pelo termo “robustez”, o conjunto das seguintes características:

- Acurácia, isto é, precisão numérica dos resultados;
- Desempenho, isto é, rapidez na obtenção dos resultados;
- Confiabilidade, isto é, bom funcionamento do algoritmo, mesmo para dados de entrada hostis ou inesperados. Em outras palavras, o algoritmo deve ser versátil;

Em particular, problemas numéricos deverão ser sanados para conferir acurácia e confiabilidade ao algoritmo a ser desenvolvido. Ao mesmo tempo, deve-se obedecer a essas exigências mantendo a

complexidade computacional do algoritmo baixa. A minimização de cálculos numéricos também é pré-requisito para a boa performance do algoritmo proposto.

Para validar nossa solução, serão apresentadas algumas demonstrações matemáticas, principalmente, da formulação teórica do algoritmo. Além disso, medidas de tempo e de precisão serão expostas, permitindo avaliar o comportamento do algoritmo em casos práticos e analisar os benefícios e deficiências da metodologia proposta.

1.3. Trabalhos Relacionados

Na literatura, uma forma bastante comum de geração de modelos de análise com o método de elementos finitos é exatamente o uso sucessivo de operações de interseção, união e diferença entre grupos de superfícies trianguladas (ARRUDA & MARTHA, 2005; PEREIRA ET AL., 2012). Tal conjunto de operadores é chamado de Operadores Booleanos, a partir do qual é possível modelar uma grande variedade de objetos. Para modelos discretos, a união de superfícies deve garantir a consistência topológica da malha unificada. Já a interseção de superfícies deve ser capaz de gerar uma boa fronteira divisória entre as superfícies em interpenetração. Isto é extremamente importante para a segunda etapa dessa operação que é, incluídos os novos vértices a cada uma das malhas, ajustar suas conectividades de forma a segregarmos duas regiões distintas em cada antiga superfície.

HOFFMANN (1992) contribuiu enormemente no desenvolvimento de algoritmos de interseção de superfícies analíticas, tanto paramétricas como implícitas. Partindo de um ponto inicial da curva de interseção, Hoffmann propõe um algoritmo para determinar os outros pontos da curva, utilizando uma Série de Taylor para estimar o próximo ponto, seguido de um Método de Newton-Raphson com passos adaptativos para correção do ponto de interseção. Hoffmann, porém, ressalta que este algoritmo falha em regiões próximas a singularidades. Hoffmann também apresenta um algoritmo para superfícies de ordem superiores, que é mais preciso e mais lento, e um algoritmo onde curvas de interseção são mapeadas parametricamente em planos.

Já em modelos discretos, LO (1995) desenvolveu um algoritmo para a interseção de superfícies, que se tornou referência básica no assunto. Ele propôs um algoritmo interseção de superfícies triangulares que automaticamente redefine as malhas dessas superfícies, adaptando-as às curvas de interseção obtidas. Os elementos (triângulos) intersectados eram excluídos e, em uma fase posterior, os buracos eram preenchidos via Avanço de Fronteira, passando a conter os pontos de interseção previamente calculados.

COELHO, GATTASS E FIGUEREDO (1999), entretanto, detectaram que, em torno de regiões de curvatura alta, os pontos de interseção calculados na técnica de LO (1995) se posicionavam fora da superfície original, o que era um resultado indesejável. Isso motivou a criação de um método para intersectar malhas de superfícies descritas por equações paramétricas, atualizando a malha das áreas com interseção e aparando os trechos de malhas fora de contato com a outra superfície. A rigor, as equações paramétricas são usadas em um método de Newton-Raphson para se determinar os pontos de interseção e, posteriormente, na avaliação desses pontos na superfície, o que permite o ajuste da malha.

SHOSTKO ET AL. (1999) também encontraram problemas de robustez no método de LO (1995), principalmente, quando aplicados a geometrias mais complexas, resultando em quinas e mudanças abruptas de direção. Assim, eles propõem um novo método para interseção de superfícies, também baseado em avanço de fronteira, que retriangula apenas os triângulos intersectados. Para ganhar robustez nas verificações de interseção aresta-triângulo, foi usada a técnica de volumes de tetraedros com sinal, proposta por AFTOSMIS ET AL. (1997).

Tempos depois, em parceria com W. X. Wang, Lo desenvolveu um novo método para se determinar curvas de interseção entre duas superfícies trianguladas e, em seguida, atualizar suas malhas (LO & WANG, 2004). Agora, os elementos intersectados passaram a ser divididos conforme a posição do ponto de interseção. É interessante que, anteriormente, LO (1995) havia recomendado um refinamento da malha nas áreas mais críticas do modelo para evitar problemas numéricos.

Desta vez, estes autores sugerem o uso da técnica de volumes de tetraedros com sinal, assim como SHOSTKO ET AL. (1999). Em um trabalho posterior, Lo e Wang apresentam novas aplicações deste mesmo método, demonstrando sua viabilidade (LO & WANG, 2005). ALVES & SILVA (2008) também comprovaram a eficácia deste método.

O algoritmo de interseção de superfícies a ser implementado nesta dissertação compreende uma rotina para busca de triângulos em interseção inédita na literatura. Ao contrário de LO & WANG (2005), esta rotina funciona também para malhas irregulares. Nesta rotina, conforme se caminha pelas malhas verificando triângulos em interseção, as curvas de interseção já são construídas com seus pontos ordenados, sem necessidade de ordenamento futuro. Além disso, o algoritmo proposto calculará interseções Segmento-Plano, utilizando Aritmética Exata não somente na classificação de interseções, mas também na determinação dos pontos de interseção, o que não é muito comum na literatura.

1.4. Organização do Texto

Esta dissertação se encontra dividida em 6 capítulos. O primeiro, correspondente à introdução, após expor a motivação e os objetivos deste trabalho, fornece um breve histórico de trabalhos relacionados à interseção de superfícies.

O segundo capítulo inicialmente comenta sobre os problemas de precisão numérica e seus impactos na modelagem. Em seguida, apresenta algumas técnicas para se suprimir tais problemas, em especial, a Aritmética Exata e os Predicados de Shewchuk que serão utilizados no algoritmo proposto.

O terceiro capítulo mostra os algoritmos que serviram de referência básica para o algoritmo proposto. Este capítulo está dividido em duas seções: uma compreendendo algoritmos de interseção de superfícies triangulares e outra abrangendo algoritmos de interseção segmento-triângulo.

O quarto capítulo explica detalhadamente o algoritmo proposto e foi decomposto em três seções: “Seleção de Candidatos a Interseção”, “Busca por Interseções” e “Interseção Segmento-Triângulo”.

O quinto capítulo mostra e discute os resultados dos testes de confiabilidade, precisão e eficiência realizados com o algoritmo proposto.

Finalmente, o sexto capítulo faz a conclusão desta dissertação, explicitando suas contribuições e deficiências e possíveis trabalhos futuros.

2

Robustez Geométrica e Aritmética Exata

2.1. Desafios de Modelos Geológicos

Em modelos geológicos, as coordenadas espaciais dos vértices das malhas geralmente apresentam ordens de grandeza muito superiores ao tamanho médio dos triângulos, principalmente se comparado aos triângulos perto de reservatórios de petróleo. Tipicamente, horizontes possuem milhares de quilômetros de extensão, enquanto a espessura de reservatórios chega a poucas dezenas de metros. Como brocas podem ficar presas em poços de petróleo por poucos centímetros, o maior erro aceitável deve ser de milímetros.

O formato de número de ponto flutuante de precisão dupla (*double*) padronizado pela IEEE (www.ieee.org) está sendo mostrado na Figura 1. Ele é composto de 1 bit indicador de sinal, 11 bits para o expoente e 52 bits para a mantissa. Portanto, *doubles* possuem precisão de apenas 16 dígitos decimais, o que pode nos levar a cometer alguns erros numéricos em nosso modelo, principalmente, por causa do arredondamento de casas decimais. Como a mantissa de um *double* é composta por 52 bits, podem ser representadas apenas 2^{52} números formados por séries de base 2. Ou seja, nem todos os números reais são representados. Alguns são truncados em seu último bit da parte decimal e outros podem ser representados por dois números vizinhos mais próximos.

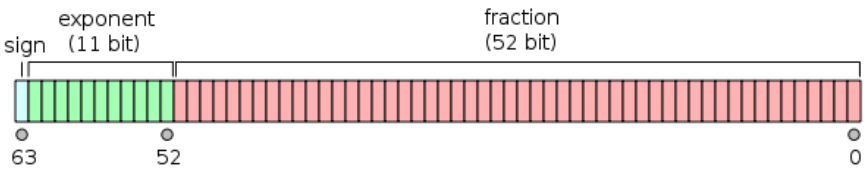


Figura 1: O formato IEEE 754 *Double Floating Point*

Principalmente dízimas periódicas (de mantissas infinitas) são difíceis de se expressar computacionalmente por causa da precisão finita. Ainda há dízimas não-períodicas que, no sistema binário, seriam representadas por dízimas periódicas. Por isso, o arredondamento de *doubles* ocorre com alguma frequência. Felizmente, o erro numérico cometido na representação de números reais normalmente é bem baixo. Entretanto, após sofrer operações aritméticas, o acúmulo de erros de arredondamento é o suficiente para fazer um teste de se "é maior ou igual a zero?" falhar, já que pode-se chegar a valores espúrios como -1e-32. Este acúmulo de erros é ainda mais perigoso quando, fugindo de controle, vence até testes de tolerância.

Essa imprecisão numérica pode nos levar tanto a obter pontos de interseção errados como ao incorreto descarte de candidatos a pontos de interseção. Como resultado, alguns pontos de interseção podem ficar mal posicionados na curva de interseção e outros, podem ficar faltando, o que causaria inconsistências topológicas no modelo.

Um estudo realizado por KETTNER ET AL. (2008) revelou problemas no uso de número de pontos flutuantes na avaliação de predicados da Geometria Computacional. Ao testar a orientação de pontos 2D sobre a reta identidade (x=y), observou-se que alguns pontos foram erroneamente classificados. Kettner manteve fixo os pontos *q* e *r* e variou o ponto *p* acrescentando 2^{-53} nas coordenadas X e Y conforme a variação de cada pixel na Figura 2.

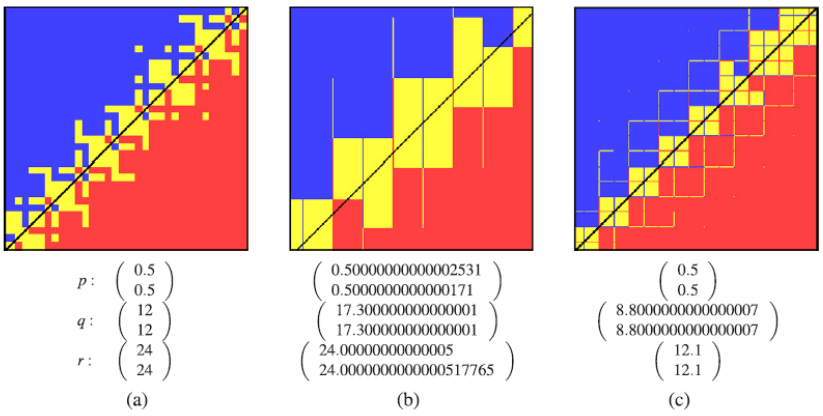


Figura 2: O estudo de KETTNER ET AL. (2008) mostrando os problemas numéricos do formato de ponto flutuante, ao orientar pontos sobre variações da reta x=y.

Fonte: KETTNER ET AL. (2008)

Esperava-se obter uma reta em 45º de pixels amarelos, representando valores zero do predicado, ou seja, pontos colineares. Os pixels azuis representam pontos acima da reta e os vermelhos, abaixo da mesma. Kettner menciona 3 tipos de problemas numéricos que podem ocorrer:

- Arredondamento para zero de números, idealmente, positivos ou negativos
- Perturbação do zero, que acabará sendo um número positivo ou negativo
- Inversão de sinal

Mesmo ao utilizar o número de precisão dupla estendido, disponível nos processadores Intel, onde utiliza-se 64 bits para a mantissa em um número de 80 bits no total, os resultados mostraram incoerências, como pode ser observado na Figura 3. Em particular, quando houve conversão do tipo *double* para o tipo *double* estendido, a avaliação do predicado ficou bem caótica, havendo muitos arredondamentos indesejados para zero. Evitando esta conversão, os resultados melhoraram, porém, ainda há erros de inversão de sinal e nenhum ponto foi considerado exatamente colinear. Kettner sugere o uso de aritmética exata para sanar os problemas encontrados.

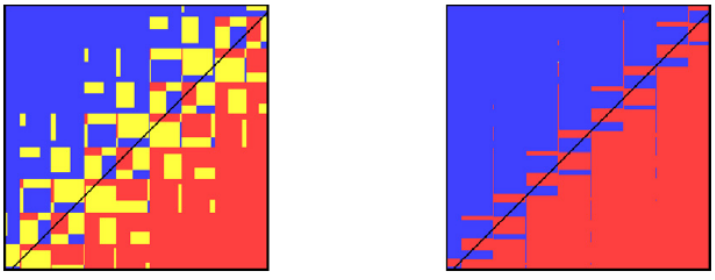


Figura 3: Reproduzindo o mesmo teste da figura 2(b) com precisão dupla estendida:

(a) Neste caso, houve conversão para números de precisão dupla;

(b) Usando somente números de precisão estendida.

Fonte: KETTNER ET AL. (2008)

2.2. Robustez na Interseção de Triângulos

Naturalmente, a busca por robustez nos algoritmos de interseção de superfícies passa pelo problema de intersectar arestas (ou segmentos de reta) com triângulos e triângulos com triângulos. Nos últimos anos, houve grandes avanços nesses estudos.

MOLLER (1997) criou um algoritmo de interseção Triângulo-Triângulo que calcula a interseção do plano de cada triângulo com o outro triângulo, obtendo dois segmentos, os quais são avaliados para produzir o segmento de interseção. Moller observou problemas de robustez nos casos onde os triângulos eram quase coplanares, o que o levou a utilizar tolerâncias.

MOLLER & TRUMBORE (1997) determinam pontos de interseção Segmento-Triângulo resolvendo um sistema de equações. Este algoritmo será mais detalhado na seção 3.2. Novamente, tolerâncias foram utilizadas.

SEGURA & FEITO (2001) desenvolveram um algoritmo rápido para testar interseções Segmento-Triângulo usando volumes com sinal. Este algoritmo se mostrou mais robusto do que o de MOLLER & TRUMBORE (1997). Entretanto, este algoritmo apenas classifica a interseção, ou seja, o ponto de interseção precisava ser calculado usando um algoritmo clássico de interseção Segmento-Plano.

JIMÉNEZ ET AL. (2009), usando volumes com sinal, criaram um algoritmo que orienta um extremo do segmento em relação ao tetraedro formado pelo outro extremo do segmento e pelos 3 vértices do triângulo. Assim, pode-se calcular as coordenadas baricêntricas no tetraedro do ponto de interseção. De acordo com os testes realizados, este algoritmo calcula interseções Segmento-Triângulo ainda mais rápido que SEGURA & FEITO (2001). Um fato curioso é que JIMÉNEZ ET AL. (2009), em busca de robustez, minimizou o número de operações aritméticas realizadas no cálculo dos volumes com sinal. Além disso, ele evitou divisões, que, entre as operações aritméticas, é uma das mais lentas e imprecisas. Mesmo assim, ele utilizou tolerâncias em suas comparações aritméticas.

Apesar de muitos autores sugerirem o uso de volumes de tetraedros com sinal para verificar e, por vezes, até classificar casos de interseções sem problemas numéricos (vide SEGURA & FEITO (2001) e JIMÉNEZ ET AL. (2009)), são raras as abordagem em que se reaproveita o valor dos volumes calculados para determinar o ponto de interseção.

Especificamente, AFTOMSIS ET AL. (1997) utilizaram Aritmética Exata Adaptativa para calcular volumes com sinal em um método de interseção Segmento-Triângulo que fora originalmente proposto por O'ROURKE (1994). Porém, AFTOMSIS ET AL. (1997) usam os valores destes volumes apenas para classificação do caso de interseção, como O'ROURKE (1994) já fazia, e não os utilizam para efetivamente calcular o ponto de interseção. De fato, isso evita o descarte incorreto de candidatos a ponto de interseção e, portanto, o problema de pontos de interseção faltando. Entretanto, os pontos de interseção criados podem continuar mal posicionados, já que os valores utilizados para determinação do parâmetro de interpolação não são totalmente confiáveis.

No início da pesquisa que originou esta dissertação, foi criado um algoritmo (vide MARQUES ET AL. (2010)) para determinar a interseção de um segmento com um triângulo que utiliza Aritmética Exata Adaptativa na verificação da interseção e reaproveita os valores dos volumes com sinal no cálculo do ponto de interseção. Contudo, este cálculo não é totalmente feito com Aritmética Exata e, mais, por utilizar Aritmética Exata Adaptativa, os pontos de interseção são calculados baseando-se em valores aproximados, logo, com pouca precisão. Outro problema é que, para verificar se um provável ponto de interseção está dentro das bordas do triângulo, é preciso calcular este ponto e, depois, verificar se ele está entre de três planos delimitadores (vide Figura 4), perpendiculares às bordas do triângulo.

Esta verificação é feita com Aritmética Exata, mesmo com planos delimitadores não sendo calculados com aritmética exata, o que pode levar à recusa de um bom ponto de interseção ou a aceitação de um mau ponto de interseção.

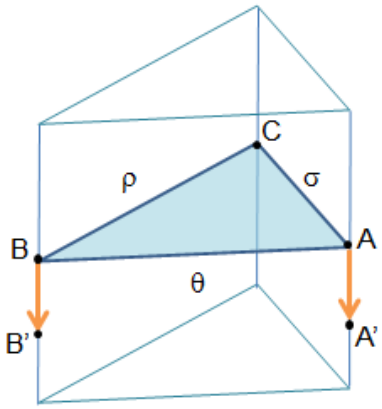


Figura 4: Planos Delimitadores do Triângulo no Algoritmo de MARQUES ET AL. (2010)

2.3. Uso de Tolerâncias

O uso de tolerâncias é um artifício bastante comum para contornar problemas de precisão. Entretanto, deve ser usado com parcimônia em regras internas de algoritmos. Quando uma regra é explícita, isto é, o usuário tem controle da tolerância utilizada, sempre é possível que o usuário refaça a operação com uma tolerância diferente até obter o efeito desejado. Este tipo de tolerância de modelagem é totalmente aceitável.

Porém, com regras implícitas, o usuário não possui controle da operação que está sendo realizada, o que pode levá-lo a situações indesejadas e incorrigíveis. Essas tolerâncias numéricas devem ser muito bem escolhidas para não se introduzir novos problemas em testes geométricos. É bastante difícil definir uma tolerância (ou um conjunto de tolerâncias) que funcionem bem em todos os casos. O desafio é achar um valor para a tolerância que não seja nem muito exigente nem muito leniente.

Para exemplificar isso, a Figura 5 mostra um caso prático. Nesta figura, o ponto de interseção ideal, que está pintado de cinza, corresponde exatamente ao vértice de uma aresta e a curva azul representa a curva de interseção esperada. Para facilitar a visualização, apenas uma das superfícies está sendo mostrada.

Suponha que, por problemas de precisão no cálculo de um ponto de interseção, ele fique deslocado, ao longo de sua aresta, para além de

sua posição ideal (no vértice). Definitivamente, ele não ficará justaposto a uma das superfícies em interseção.

Se a tolerância utilizada for muito permissiva, este ponto de interseção será incluído na curva de interseção, o que resultará em uma curva que não honrará bem a interseção com uma das superfícies (senão com ambas). Esta curva está pintada de vermelho na Figura 5.

Por outro lado, uma tolerância muito severa poderia descartar tal ponto de interseção. Dessa forma, a curva de interseção possuirá um trecho que não estará sobreposto a nenhuma das superfícies em interseção, como mostrado pela curva verde da Figura 5.

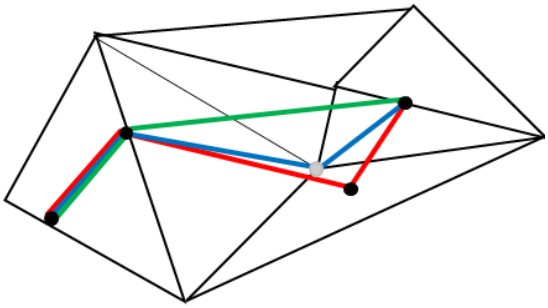


Figura 5: Problemas de Tolerância

A curva azul representa a curva ideal, passando pelo ponto cinza.
A curva vermelha tem um ponto de interseção deslocado e
a curva verde descartou um ponto de interseção.

No exemplo mostrado, uma boa tolerância seria capaz de corrigir o ponto de interseção, atraindo-o para o vértice. Uma tolerância fixa no código (*hard-coded*) seria, a princípio, uma solução. Porém, os tamanhos das superfícies variam muito de um modelo para outro e, às vezes, até dentro do mesmo modelo. Uma tolerância muito baixa pode não funcionar bem em modelos grandes, pois ignoraria casos onde se deveria aproximar pontos de interseção para as extremidades do segmento. Ao mesmo tempo, uma tolerância baixa seria fundamental em modelos pequenos, onde precisão é mais importante e mais difícil de se obter. Ao contrário, se definirmos uma tolerância alta, ela vai funcionar bem em modelos grandes, mas, nos modelos pequenos, os pontos de interseção serão sempre atraídos para as extremidades dos segmentos.

Para resolver isso, uma tolerância baseada no tamanho da menor aresta da malha da superfície em questão poderia ser utilizada. Contudo, se nossa malha triangular possuir triângulos de tamanhos altamente discrepantes, nossa tolerância pode ser exigente demais para alguns pares aresta-triângulo e muito branda para outros pares.

Mesmo utilizando-se o tamanho da aresta corrente como referência, é necessário calibrar a tolerância utilizada: 1% deste valor, 0,1%, 10% ? Novamente, esta calibração não é muito óbvia. E pode acabar atraindo indevidamente pontos no meio do segmento para vértices do segmento.

Não obstante, há um problema ainda maior. Tolerâncias só podem ser utilizadas definindo-se regras absolutas. Por exemplo, para a atração dos pontos de interseção para vértices de segmentos, não é possível formular uma regra genérica de correção para pontos de interseção que devem se localizar no meio de segmentos. Em outras palavras, não há como definir uma ação “atraia para o ponto correto” (o que é relativo) de forma que não haja novamente erros numéricos.

Uma saída para isso seria o refinamento progressivo do erro numérico, por meio de um processo iterativo, o que é normalmente caro. De qualquer forma, utilizar tolerâncias não será eficaz em nosso algoritmo.

2.4. Tecnologias para Robustez Geométrica

Uma alternativa a tolerâncias é o uso de técnicas de controle de erro numérico, como Aritmética Intervalar (MOORE 1979), Aritmética Afim (FIGUEIREDO & STOLFI 2004) e Aritmética Exata (PRIEST 1991).

Na Aritmética Intervalar, usa-se um intervalo real compacto como objeto básico. Um número real pode ser representado como um intervalo onde inicialmente ele mesmo é o ínfimo (valor mínimo) e o supremo (valor máximo) do intervalo. Conforme operações aritméticas, como adição, subtração, multiplicação e divisão, são realizadas sobre os intervalos (separadamente sobre seu ínfimo e supremo), o intervalo tende a se expandir em torno do resultado da operação, concentrando, assim, a maior faixa de erro possível de ser cometido com o valor original. Dessa

forma, os testes de comparação são realizados respeitando este intervalo e, portanto, tornam-se mais confiáveis. Isto seria bastante útil para não classificarmos casos de interseção erroneamente. Porém, não nos ajudaria no cálculo do ponto de interseção, já que teríamos que escolher qual valor dentro do intervalo obtido deve ser utilizado. Na prática, isso significa que teríamos vários candidatos a um mesmo ponto de interseção, muito próximos entre si. A escolha de seu melhor representante parece não ser muito óbvia.

Já a Aritmética Afim é uma técnica que surgiu para corrigir o problema de dependência que ocorre na Aritmética Intervalar. Este problema causa um aumento exagerado no tamanho dos intervalos, o que é indesejado. Na aritmética afim, atribui-se, a todo número real, ruídos em torno de um valor central, ponderados por desvios parciais. Assim, números reais são expressos por intervalos calculados sob a forma de somatórios. Semelhantemente à aritmética intervalar, operadores aritméticos são redefinidos para adequarem-se a essa nova representação dos números reais. Aritmética Afim é muito usada na interseção entre superfícies implícitas e paramétricas.

Além dessas ferramentas existe a Aritmética Exata que, por ser a técnica adotada no algoritmo proposto, será descrita em mais detalhes na seção a seguir.

2.5. Aritmética Exata

A Aritmética Exata é uma técnica originalmente desenvolvida por PRIEST (1991) onde todos os números de ponto flutuante, ao passar por operações aritméticas, são quebrados em duas componentes de números de pontos flutuantes cujas representações binárias são não-sobrejacentes (*non-overlaping*). Isto significa que, essas componentes têm diferentes ordens de grandeza e, quando somados, não gerarão “vai-um” (*carry*). Deve-se lembrar que, com números binários, só ocorre “vai-um” quando se adiciona, na mesma casa, uma unidade com uma unidade.

Soma de números com *Overlapping*	Soma de números sem *Overlapping*
1 1	
1.10**1**0**1**0000000	1.101010000000
+0.00**1**0**1**0000000	+0.000000010100
1.110100000000	1.101010010100

Figura 6: Soma de Números Binários com e sem *Overlapping*

Repare, na Figura 6, que apenas números com *overlapping*, produzem "vai-um", simbolizados pelos pequenos números '1' acima do primeiro termo da soma.

Ao se quebrar um número em componentes, normalmente, uma componente possui ordem de grandeza baixa (sendo tipicamente associada ao erro numérico cometido até então) e outra possui ordem de grandeza mais alta, representando uma aproximação do resultado obtido. Essas componentes são obtidas realizando provas reais das operações realizadas. Por exemplo, suponha uma soma de números reais,

$$a + b = c$$

Logo, idealmente

$$c - b = a$$
$$c - a = b$$

Entretanto, na prática,

$$c - b = a + \varepsilon_a$$
$$c - \mathrm{a} = b + \varepsilon_b$$

onde ε_a e ε_b representam os erros numéricos associados respectivamente aos números a e b. Portanto, pode-se determinar o erro ε_c, cometido na soma de a com b, da seguinte forma:

$$a + b = c$$
$$b_{virtual} = c - a$$
$$\varepsilon_c = b_{virtual} - b$$

onde $b_{virtual}$ representa uma estimativa de b contendo o erro numérico cometido na soma de a com b.

Assim, a soma de a com b resultou em duas componentes sem *overlapping*, c e ε_c. O simples fato de se utilizar componentes permite evitar erros de arredondamento causados por deslocamentos (*shifts),* realizados internamente (pela CPU) nessas operações aritméticas para equiparar números de ordens de grandezas diferentes e, então, somá-los casa por casa binária. A Figura 7 mostra como a soma de números de ordens de grandezas diferentes pode resultar em um número truncado, ou seja, com erro de arredondamento.

Soma de números com Ordens de Grandeza Semelhantes	Soma de números com Ordens de Grandeza Discrepantes
1.10101e+00	1.10101e+32
+1.01000e -02	+1.01000e -32
Fazendo os *shifts:*	Fazendo os *shifts:*
1.101010000000	1.101010...00000
+0.001010000000	+0.000000...01010
1.110100000000	1.101010...**00000**

Figura 7: Soma de Números Binários de Ordens de Grandeza Semelhantes e Discrepantes

Na medida em que outras operações aritméticas são realizadas, essas componentes são novamente quebradas, obtendo componentes de ordem de grandezas intermediárias. Ao final, podemos unir todas as componentes obtidas, estimando o erro numérico acumulado. Se este erro numérico acumulado for menor do que a precisão máxima da

máquina, ele pode ser descartado. Senão, o valor obtido é corrigido, levando em conta esse erro. Assim, o acúmulo de erros de arredondamento é evitado, o que é especialmente perigoso em cálculos de valores usados em testes geométricos, principalmente os que envolvem números muito grandes e muito pequenos.

A Figura 8 mostra a árvore de operações de Aritmética Exata no cálculo da expressão

$$(x_1 + y_1)^2 + (x_2 + y_2)^2$$

equivalente a

$$x_1^2 + x_2^2 + 2x_1y_1 + 2x_2y_2 + y_1^2 + y_2^2$$

Nesta figura, o operador x circulado (⊗) representa a multiplicação com Aritmética Exata, que recebe dois números de tipo *double* e devolve sempre um número com duas componentes. Já o operador + com envoltória quadrada (⊞) representa a soma de componentes, recebendo um número com componentes e devolvendo um único número com mais componentes.

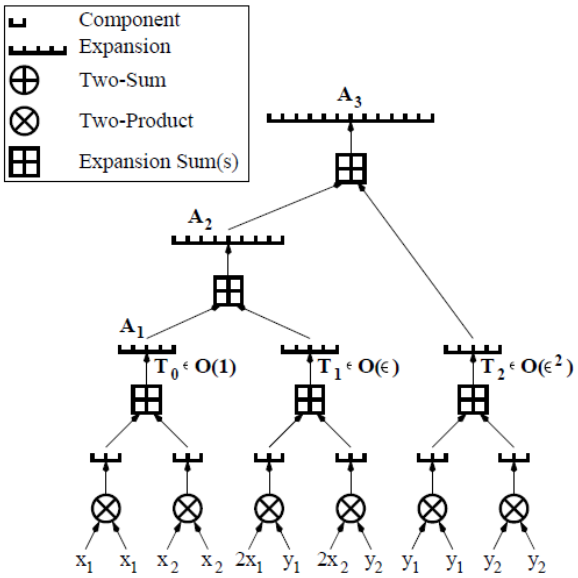


Figura 8: Árvore de Operações de Aritmética Exata

Fonte: SHEWCHUCK (1996a,b)

Isso levou Fortune e Wyk, a construir os primeiros predicados de geometria computacional utilizando Aritmética Exata (FORTUNE & WYK, 1993). Entretanto, foi observado que a Aritmética Exata ocasiona grande perda de performance, o que foi contornado utilizando aritmética de multi-precisão para números inteiros e filtros de números de pontos flutuante baseados em análise de intervalos. HOFFMANN (1992) também afirma que, apesar de a Aritmética Exata solucionar o problema de precisão, ela é muito lenta.

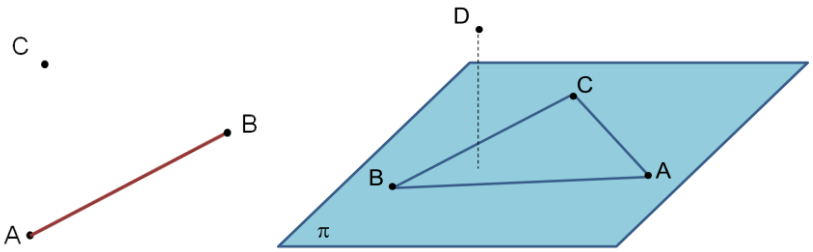
Pouco tempo depois, Shewchuk introduziu adaptatividade na Aritmética Exata para torná-la mais rápida. Na Aritmética Exata Adaptativa (SHEWCHUCK 1996), em vez de se unir todas as componentes somente no final de todas as operações aritméticas, as componentes de ordens de grandeza semelhantes passaram a ser unidas ao mesmo tempo em que operações aritméticas são realizadas. Assim, é possível estimar o erro cometido antes de se terminar todo o cálculo a ser realizado. E, principalmente, garante-se que, a partir de certo momento, não é mais necessário controlar o erro numérico, pois ele não será mais capaz de adulterar o resultado final do cálculo. Na prática verificou-se que a Aritmética Exata Adaptativa possui um custo médio muito menor, já que tende a ser mais cara apenas quando uma maior precisão é realmente necessária. Quando este não é o caso, ela é apenas ligeiramente mais cara que a Aritmética Tradicional.

Por eficiência, utilizaremos Aritmética Exata Adaptativa para classificar o caso de interseção (e possivelmente descartá-la) e, se confirmada a interseção, utilizaremos Aritmética Exata Não-Adaptativa para calcularmos os pontos de interseção. Assim, uma das metas deste trabalho será avaliar se a Aritmética Exata Adaptativa aumenta a segurança no descarte de pontos de interseção. Outra é verificar se a Aritmética Exata Não-Adaptativa, utilizada a partir de seus predicados, é capaz de calcular os pontos de interseção com maior precisão.

2.6. Predicados de Shewchuk

SHEWCHUCK (1996) também implementou alguns predicados comuns da Geometria Computacional utilizando Aritmética Exata Adaptativa. Para efeito de comparação, ele também implementou versões em Aritmética Exata Não-Adaptativa e Aritmética Tradicional. São esses predicados:

- *orient2d*: determina se um ponto no espaço 2D se localiza acima, abaixo ou exatamente sobre uma reta definida por dois pontos (Figura 9(a)).
- *orient3d*: determina se um ponto no espaço 3D se localiza acima, abaixo ou exatamente sobre um plano definido por 3 pontos (Figura 9(b))
- *incircle*: determina se um ponto no espaço 2D se localiza no interior, exterior ou exatamente sobre uma circunferência definida por 3 pontos.
- *insphere*: determina se um ponto no espaço 3D se localiza no interior, exterior ou exatamente sobre uma esfera definida por 4 pontos.



(a) **(b)**

Figura 9: Os Predicados de Shewchuk *orient2d* e *orient3d*

(a) Predicado *orient2d* fornece a orientação de um ponto em relação a uma reta
(b) Predicado *orient3d* fornece a orientação de um ponto em relação a um plano

Estes predicados são calculados da seguinte forma:

$$orient2d(A,B,C) = \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} = \begin{vmatrix} A_x - C_x & A_y - C_y \\ B_x - C_x & B_y - C_y \end{vmatrix}$$

(1)

$$orient3d(A,B,C,D) = \begin{vmatrix} A_x & A_y & A_z & 1 \\ B_x & B_y & B_z & 1 \\ C_x & C_y & C_z & 1 \\ D_x & D_y & D_z & 1 \end{vmatrix} = \begin{vmatrix} A_x - D_x & A_y - D_y & A_z - D_z \\ B_x - D_x & B_y - D_y & B_z - D_z \\ C_x - D_x & C_y - D_y & C_z - D_z \end{vmatrix}$$

(2)

$$incircle(A,B,C,D) = \begin{vmatrix} A_x & A_y & A_x^2 + A_y^2 & 1 \\ B_x & B_y & B_x^2 + B_x^2 & 1 \\ C_x & C_y & C_x^2 + C_x^2 & 1 \\ D_x & D_y & D_x^2 + D_x^2 & 1 \end{vmatrix}$$

$$= \begin{vmatrix} A_x - D_x & A_y - D_y & (A_x - D_x)^2 + \left(A_y - D_y\right)^2 \\ B_x - D_x & B_y - D_y & (B_x - D_x)^2 + \left(B_y - D_y\right)^2 \\ C_x - D_x & C_y - D_y & (C_x - D_x)^2 + \left(C_y - D_y\right)^2 \end{vmatrix}$$

(3)

$$insphere(A,B,C,D,E) = \begin{vmatrix} A_x & A_y & A_z & A_x^2 + A_y^2 + A_z^2 & 1 \\ B_x & B_y & B_z & B_x^2 + B_y^2 + B_z^2 & 1 \\ C_x & C_y & C_z & C_x^2 + C_y^2 + C_z^2 & 1 \\ D_x & D_y & D_z & D_x^2 + D_y^2 + D_z^2 & 1 \\ E_x & E_y & E_z & E_x^2 + E_y^2 + E_z^2 & 1 \end{vmatrix} =$$

$$= \begin{vmatrix} A_x - E_x & A_y - E_y & A_z - E_z & (A_x - E_x)^2 + \left(A_y - E_y\right)^2 + (A_z - E_z)^2 \\ B_x - E_x & B_y - E_y & B_z - E_z & (B_x - E_x)^2 + \left(B_y - E_y\right)^2 + (B_z - E_z)^2 \\ C_x - E_x & C_y - E_y & C_z - E_z & (C_x - E_x)^2 + \left(C_y - E_y\right)^2 + (C_z - E_z)^2 \\ D_x - E_x & D_y - E_y & D_z - E_z & (D_x - E_x)^2 + \left(D_y - E_y\right)^2 + (D_z - E_z)^2 \end{vmatrix}$$

(4)

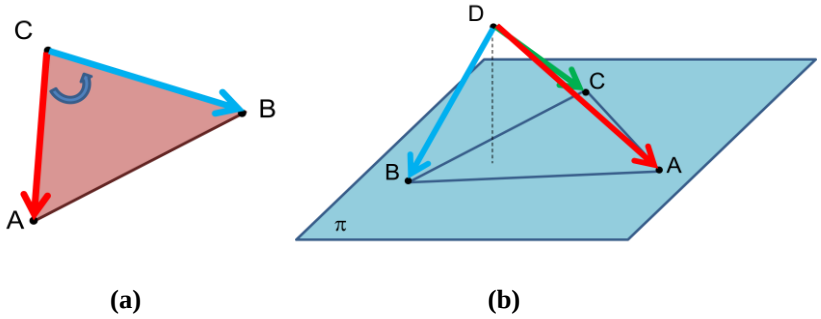
Implementar esses simples predicados com Aritmética Exata Adaptativa já é algo bem trabalhoso e, normalmente, o código produzido é de difícil compreensão, principalmente após otimizações. Por este motivo, a Aritmética Exata será introduzida no algoritmo proposto por meio desses predicados em vez de ser incluída em cada operação aritmética. É bem verdade que a inclusão de Aritmética Exata operação por operação permite otimizações e certo ganho de desempenho do algoritmo. Contudo, isto tornaria o código mais suscetível a *bugs*, visto que esta não seria uma implementação trivial.

Assim, será muito importante realizarmos diversas operações vetoriais por meio dos predicados de Shewchuk. Entretanto, somente após entendermos os valores de retorno desses predicados que poderemos de fato utilizá-los no algoritmo proposto. A seguir, investigaremos o significado dos valores retornados pelos predicados *orient2d* e *orient3d*.

Sabe-se que, pelos determinantes das Equações (1) e (2), estes predicados são equivalentes a um produto vetorial e um produto triplo escalar (ou produto misto), como demonstrado pelas Equações (5) e (6). Tais produtos podem ser geometricamente visualizados na Figura 10.

$$orient2d(A,B,C) = (A-C)\times(B-C) = \overrightarrow{CA}\times\overrightarrow{CB} = \overrightarrow{AB}\times\overrightarrow{AC} \tag{5}$$

$$\begin{aligned} orient3d(A,B,C,D) &= (A-D)\cdot\big((B-D)\times(C-D)\big) \\ &= \overrightarrow{DA}\cdot\big(\overrightarrow{DB}\times\overrightarrow{DC}\big) = -\Big(\overrightarrow{AD}\cdot\big(\overrightarrow{AB}\times\overrightarrow{AC}\big)\Big) \end{aligned} \tag{6}$$



(a) **(b)**

Figura 10: Os Predicados de Shewchuk *orient2d(a)* e *orient3d(b)* respectivamente como um produto vetorial e um produto misto.

Assim, pelas propriedades do produto vetorial, o valor retornado por *orient2d* equivale a área com sinal do paralelogramo formado pelos vetores CA e CB. Ou seja, o predicado *orient2d* retorna o dobro da área com sinal do triângulo ABC. Adicionalmente, se os pontos A, B e C estiverem orientados em sentido anti-horário, o sinal retornado pelo predicado será sempre positivo e se estiverem em sentido horário, será negativo. O predicado retornará zero somente se o ponto C for colinear a reta AB. Dessa forma, é possível saber a orientação do ponto C em relação a reta AB. Por exemplo, para A à esquerda de B,

$$orient2d(A,B,C)\begin{cases} > 0 \ \leftrightarrow C\ está\ acima\ da\ reta\ dada\ pelos\ pontos\ A\ e\ B \\ = 0 \ \leftrightarrow C\ pertence\ a\ reta\ dada\ pelos\ pontos\ A\ e\ B \\ < 0 \ \leftrightarrow C\ está\ abaixo\ da\ reta\ dada\ pelos\ pontos\ A\ e\ B \end{cases} \quad (7)$$

Se A não estiver a esquerda de B, basta utilizar a Equação (7) com seus sinais invertidos.

Já pelas propriedades do produto misto, o módulo de *orient3d* equivale ao volume com sinal do paralelepípedo formado pelos vetores DA, DB e DC. Dessa forma, o predicado orient3d retorna 6 vezes o volume com sinal do tetraedro ABCD. Se os pontos A,B e C estiverem orientados no sentido anti-horário, o predicado será positivo somente se o ponto D estiver abaixo do plano dado pelos pontos A,B e C. Ainda com A,B e C no sentido anti-horário, o predicado será negativo se D estiver acima desse plano e zero, se D for coplanar. Assim, pode-se descobrir a orientação do ponto D em relação ao plano dado pelos pontos A, B e C no sentido anti-horário da seguinte forma:

$$orient3d(A,B,C,D)\begin{cases} < 0 \ \leftrightarrow D\ está\ acima\ do\ plano\ dado\ pelos\ pontos\ A,B\ e\ C \\ = 0 \ \leftrightarrow D\ pertence\ ao\ plano\ dado\ pelos\ pontos\ A,B\ e\ C \\ > 0 \ \leftrightarrow D\ está\ abaixo\ do\ plano\ dado\ pelos\ pontos\ A,B\ e\ C \end{cases} \quad (8)$$

Se os pontos, A, B e C estiverem orientados no sentido horário, os sinais acima se invertem. De qualquer forma, de posse dos predicados *orient2d* e *orient3d* é possível saber a orientação de um ponto relativa a uma reta ou um plano com base apenas no sinal desses predicados.

É interessante ressaltar algumas outras propriedades do produto misto que serão utilizadas ao longo desta dissertação:

- Troca de Ponto de Referência:

$$orient3d(A,B,C,D) = orient3d(B,C,D,A)$$

- Inversão do Sentido de um dos vetores:

$$(A-D)\cdot(B-D)\times(C-D) = -(D-A)\cdot(B-D)\times(C-D)$$

- Inversão da Ordem de Vetores Consecutivos:

$$orient3d(A,B,C,D) = -orient3d(A,C,B,D)$$

- Conjunto de Vetores Linearmente Dependentes

$$orient3d(A, B, A, D) = 0$$

Todas estas propriedades são visíveis geometricamente. O produto vetorial também dispõe destas mesmas propriedades. Em particular, um artifício algébrico a ser amplamente utilizado será que um número par de inversões não altera o sinal do produto misto.

2.7. Operação de Divisão

Para obtermos o ponto de interseção, precisaremos utilizar divisão. As outras três operações aritméticas básicas (soma, subtração e multiplicação) podem ser obtidas através dos Predicados de Shewchuk. MÖLLER (1997) implementou um algoritmo sem operações de divisão capaz de determinar se há interseção segmento-triângulo, porém, também sem calcular pontos de interseção. Apesar de PRIEST (1991) ter formulado a operação de divisão com Aritmética Exata, não há implementações disponíveis.

Foi cogitado realizar uma abordagem híbrida, utilizando Aritmética Intervalar somente para efetuar a divisão. Entretanto, os testes preliminares realizados mostraram que a ordem lexicográfica é suficiente para garantir a comutatividade da operação de interseção segmento-triângulo.

Na ordenação lexicográfica, se ordena pontos comparando suas coordenadas X, Y e Z. Após estabecelida esta ordem, operações serão realizadas sobre estes pontos sempre respeitando esta ordem. Assim, na verdade, são evitados os efeitos adversos da troca de ordem destes pontos. Esta é uma solução trivial e de baixo custo computacional para o problema de comutatividade.

Em particular, a interseção de dois cubos levemente rotacionados produziu curvas de interseção fechadas (com pontos de interseção iguais) apenas utilizando ordenação lexicográfica. Sem tal ordenação, as curvas não fecharam, mesmo quando se utilizou Aritmética Exata. Nesse caso, foi possível observar que a Aritmética Exata sem a ordenação topológica

garantiu maior precisão aos pontos de interseção, mas não os tornou rigorosamente iguais. A ordenação lexicográfica sem aritmética exata também não foi capaz de fechar as curvas de interseção. Portanto, é necessário que tanto a Aritmética Exata quanto a Ordenação Lexicográfica estejam presentes.

É importante ressaltar, porém, que utilizando a ordenação lexicográfica, nossa divisão continua inexata e, possivelmente, adulterará o resultado. Inclusive, porque a divisão é uma das operações mais imprecisas das realizadas por Unidades Lógico-Aritméticas (em inglês, ALU). Pelo menos, com a Aritmética Exata, o caso de interseção não será erroneamente avaliado e, com a Ordenação Lexicográfica, o ponto de interseção será sempre o mesmo, ainda que se inverta a orientação dos segmentos e triângulos.

3

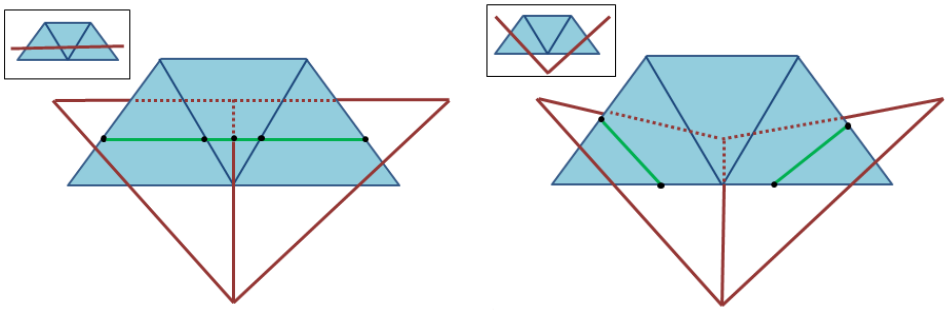
Algoritmos de Interseção

3.1. Interseção de Superfícies Triangulares

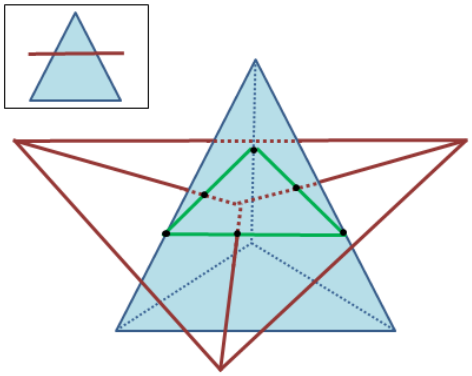
Na interseção de superfícies discretas, não basta apenas calcularmos pontos de interseção. É fundamental colocarmos os pontos de interseção em uma sequência que defina curvas de interseção do tipo curva poligonal (*polyline*). Isto é muito importante para o modelador, já que são elas que, topologicamente, delimitam superfícies.

Contudo, é relativamente comum duas superfícies em interseção possuírem mais de uma curva de interseção. Nesse caso, é preciso determinar a qual curva cada ponto pertence, além de estabelecer a ordem interna dos pontos de cada curva.

As curvas de interseção também não precisam necessariamente ser curvas poligonais abertas. Ou seja, podem ser curvas poligonais fechadas. Em particular, LO & WANG (2004) chamam essas curvas de *loops* (laços) e as poligonais abertas de *chains* (cadeias abertas). A Figura 11 exibe três diferentes possibilidades de interseção: uma simples curva do tipo cadeia aberta; mais de uma curva do tipo cadeia aberta; uma curva do tipo *loop*.



(a) **(b)**



(c)

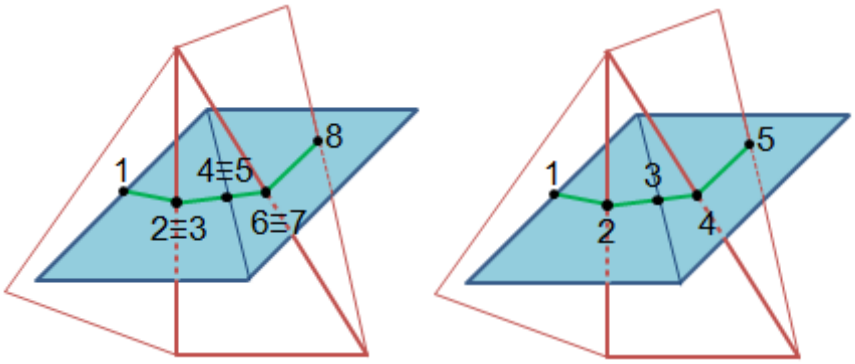
Figura 11: Várias Possibilidades de Interseção

(a)Interseção com uma única curva do tipo *chain*; (b)Interseção com duas curvas do tipo *chain*; (c)Interseção com uma curva do tipo *loop;*

Deve ser ressaltado que, há ainda outras possibilidades que não foram mostradas nesta figura: vários loops, vários loops e uma cadeia aberta, vários loops e várias cadeias abertas. A quantidade das curvas de interseção e o tipo de cada uma dessas curvas dependem somente da geometria das superfícies consideradas. Por isso, há tantas possibilidades. Um algoritmo robusto deve ser capaz de lidar com todas essas possibilidades.

De fato, a estratégia de ordenação de curvas desempenha um papel central, determinando inclusive o tipo de entidades geométricas de nível mais baixo que vamos intersectar: triângulos com triângulos ou arestas com triângulos. Utilizando interseções triângulo-triângulo, todos os pontos da curva de interseção, com exceção do primeiro e do último ponto de curvas abertas, apareceriam duplicados no vetor de pontos de interseção. O mesmo não ocorre utilizando interseções aresta-triângulo (com exceção do raro caso onde uma aresta intersecta um triângulo exatamente na sua aresta de borda). Assim, o esforço gasto no cálculo de interseções triângulo-triângulo é, em média, o dobro do gasto em interseções segmento-triângulo.

A Figura 12 mostra exatamente isto. Note que com interseções triângulo-triângulo foram obtidos 8 pontos de interseção (a maioria duplicados), enquanto que com interseções segmento-triângulo, foram obtidos apenas 5 pontos de interseção, nenhum duplicado.



(a) **(b)**

Figura 12: Pontos Duplicados na Interseção Triângulo-Triângulo(a) não são duplicados na Interseção Segmento-Triângulo(b)

Por outro lado, a interseção entre triângulos geralmente retorna dois pontos de interseção. Assim, têm-se pontos de interseção ordenados em pares. Ou seja, nosso esforço para ordenar pontos de interseção cai à metade em comparação com a ordenação necessária a pontos de interseção segmento-triângulo.

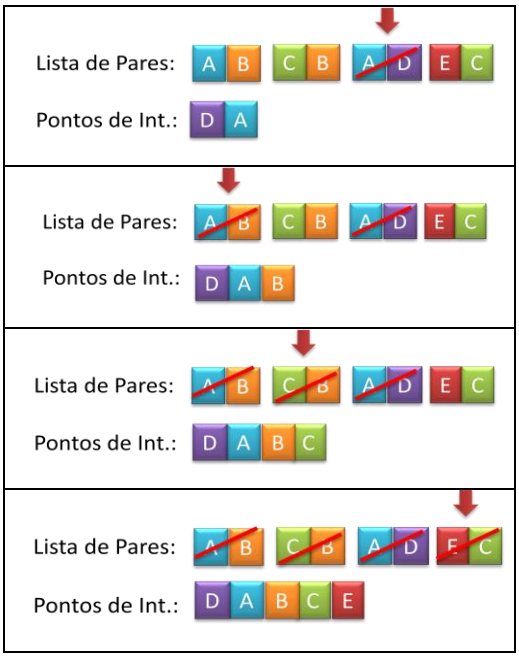
Outra vantagem do fato de utilizarmos interseções triângulo-triângulo é que podemos ordenar pares de pontos de interseção com base nos pontos duplicados. O Algoritmo Força Bruta, mostrado na próxima sub-seção, faz exatamente isso.

3.1.1. Algoritmo “Força Bruta”

Um algoritmo “força bruta” para a interseção de superfícies começa intersectando todos os triângulos com todos os triângulos, obtendo uma lista de pares de pontos de interseção. Dois pares de pontos que possuem um ponto em comum representam segmentos consecutivos na curva de interseção. Assim, um algoritmo de ordenação de tempo quadrático pode ser criado. A Figura 13 mostra brevemente o pseudocódigo e o passo-a-passo deste algoritmo. Não convém explicá-lo em mais detalhes. Só é importante advertir que, da forma como está exposto na Figura 13, este algoritmo trata apenas uma única *chain*. Contudo, ele pode facilmente ser estendido para tratar múltiplas curvas de interseção, incluindo vários *chains* e *loops*.

```
Sort_Chain_Pairs:
Find a pair g containing a point without repetition P
Mark g as visited
Set Q as the point in the pair g which is not P
Include P in the intersection curve
Include Q in the intersection curve
While there is another unvisited pair h containing Q do
    Mark h as visited
    Set P as the point in the pair h which is not Q
    Include P in the intersection curve
    Set Q as the point P
End While
```

(a)



(b)

Figura 13: Algoritmo de Tempo Quadrático para Ordenação de Pares de Pontos de Interseção Triângulo-Triângulo (a) e seu Passo-a-Passo (b).

Deve ser ressaltado, porém, que erros numéricos podem fazer com que dois pontos idealmente iguais sejam diferentes. Por isso, é comum utilizar-se filtros topológicos baseados em tolerâncias para determinar se dois pontos representam o mesmo ponto. Entretanto, de posse de um algoritmo robusto de interseção triângulo-triângulo, o problema não ocorre, já que serão obtidos pontos rigorosamente iguais.

De qualquer forma, para malhas extensas, é inviável sair intersectando todos os triângulos com todos os triângulos ou todas as arestas com todos os triângulos e, em seguida, ordenar os pontos de interseção.

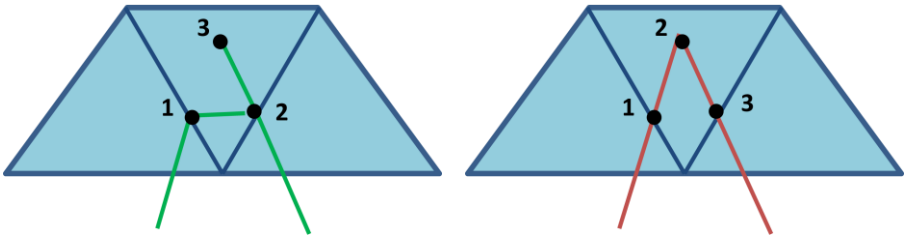
3.1.2. Algoritmo de LO

Intersectar todas as arestas de uma malha contra todos os triângulos de outra malha é uma operação muito cara. Mesmo assim, LO (1995) adotou a estratégia de calcular interseções entre todos os pares de triângulos com um método que, a fundo, consistia em 6 verificações de interseção aresta-triângulo. Depois, em uma etapa posterior, os pontos de interseção eram ordenados.

Fazendo isso, a etapa do cálculo de interseções teria uma complexidade computacional de $\theta(e_A \cdot t_B + e_B \cdot t_A)$, sendo e_k o número de arestas da malha K e t_k o número de triângulos da malha K. Como em uma malha triangular, cada triângulo possui três arestas, geralmente compartilhadas com outros três triângulos (há raras exceções que são arestas de borda), podemos afirmar que $e \approx 3 \cdot t/2$. Assim, nossa complexidade fica $O(t_A \cdot t_B + t_B \cdot t_A)$, o que é equivalente a $O(t_A \cdot t_B)$. Logo, podemos dizer que nosso algoritmo tem ordem quadrática, ou seja, é $O(n^2)$, se tomarmos n pelo o número total de triângulos em ambas as malhas, isto é, $n=t_A+t_B$.

Para ordenar todos os pontos de interseção obtidos, LO (1995) minimizou as distâncias euclidianas entre cada par de dois pontos consecutivos, uma vez que suas curvas de interseção são contínuas e suaves. O custo da etapa de ordenação, então, é $O(m^2)$, sendo m o número de pontos de interseção.

Esta abordagem, porém, pode introduzir erros de ordenação para geometrias mais complexas. Observe, na Figura 14, que nem sempre pontos de maior proximidade são naturais sucessores em curvas de interseção. Não por acaso, testes realizados por SHOSTKO ET AL. (1999) detectaram problemas na ordenação de pontos.



(a) **(b)**

Figura 14: Erro de Ordenação de Lo (1995)

(a) Curva de interseção formada pelos pontos de maior proximidade;

(b) Curva de Interseção corretamente ordenada

Do ponto de vista de desempenho, o grande gargalo deste algoritmo é o cálculo dos pontos de interseção. Assim, Lo propôs, em parceria com Wang, um novo algoritmo mais eficiente e sem os problemas citados.

3.1.3. Algoritmo de LO & WANG

Neste algoritmo, LO & WANG (2004) constroem um *grid* (grade) regular para descartar cedo triângulos garantidamente sem interseção (*background grid filter*). Para se criar esse filtro, primeiro, marca-se quais triângulos pertencem a quais células. Um triângulo que está apenas parcialmente contido em uma célula deve pertencer a esta célula também. Após criado esse filtro, triângulos que pertencem a células diferentes certamente não estarão em interseção. Na prática, um triângulo só poderá ter interseção com os triângulos pertencentes às células que o contém.

Assim, já se reduz bastante o número de verificações de interseção triângulo-triângulo. Entretanto, adicionam-se verificações de continência parcial de triângulos com células cúbicas, que são bem mais baratas, utilizando, por exemplo, o algoritmo proposto por MÖLLER (2001).

A Figura 15 mostra algumas células em um modelo com três superfícies sem interseção. Como não há interseção, cada célula contém apenas triângulos de uma mesma superfície ou está vazia. Caso

houvesse interseção, algumas células possuiriam triângulos de superfícies diferentes em seu interior.

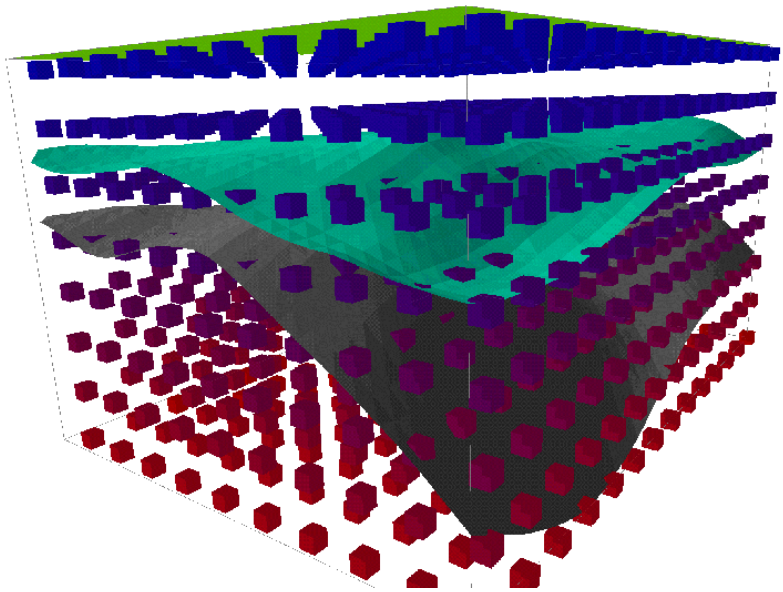


Figura 15: Background Grid Filter Regular de um Modelo sem Interseção

Porém, este *grid* foi concebido para ser usado em malhas regulares e suaves, o que não é o caso de objetos naturais. Como nossas superfícies são representadas por malhas irregulares, o filtro proposto não irá funcionar adequadamente, ainda que haja refinamento progressivo de nossas superfícies. Um *grid* regular com células grandes não iria filtrar bem interseções de triângulos pequenos. Por outro lado, um *grid* regular cujo tamanho das células fosse, no máximo, igual ao menor dos triângulos de ambas as superfícies, seria tão refinado que teria pouco efeito nos triângulos maiores. Dessa forma, a utilização de um grid regular não seria muito eficiente.

Já o uso de estruturas espaciais hierárquicas como uma Octree (MEAGER 1980), uma R-Tree (GUTTMAN 1984), uma R+Tree (SELLIS ET AL. 1987) ou uma R*-Tree (BECMANN ET AL. 1990), em contrapartida, seria bem mais apropriado. Nesse caso, no último nível da árvore, todas as células seriam caixas envolventes (*bounding boxes)* de triângulos.

A Figura 16 mostra um exemplo de Octree. Na raiz desta árvore, está a célula que contém todo o modelo. Já no nível 1, cada célula representa um octante do modelo e contém todas as células dentro deste octante. No nível 2, cada célula representa um octante de sua célula-pai e, novamente contém todas as sub-células dentro do referido octante. E assim sucessivamente até que, no último nível, cada célula contém todos os triângulos os quais encobre (ainda que parcialmente).

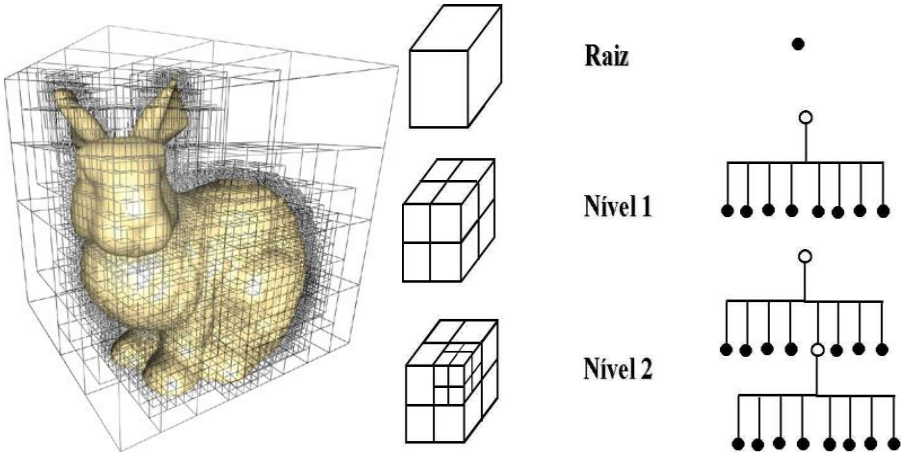


Figura 16: Octree com diferentes níveis de refinamento

LO & WANG (2004) também criaram um método, chamado de TNOIT (*Tracing Neighbours of Intersecting Triangles* ou, em tradução livre, navegação por vizinhos de triângulos intersectados), onde dependendo do caso de interseção Segmento-Triângulo, eles partem para um teste de entidades em interseção específico. A saber, quando um ponto de interseção localiza-se no interior de um triângulo, a busca continua mantendo este triângulo. Quando o ponto de interseção localiza-se em uma aresta do triângulo, a busca deve continuar no triângulo que compartilha esta aresta. E, finalmente, se o ponto de interseção localiza-se em um vértice do triângulo deve-se prosseguir com a busca em todos os triângulos que compartilham esse vértice. A Figura 17 mostra como o TNOIT prossegue em alguns casos particulares.

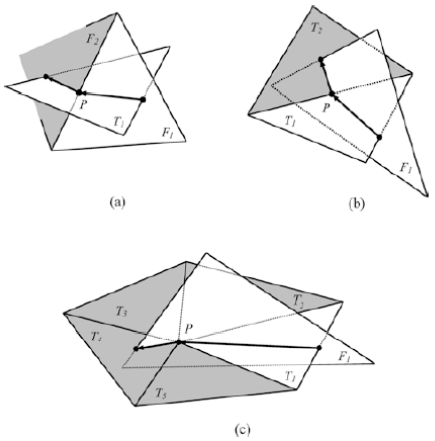


Figura 17: Construção da Curva de Interseção com o TNOIT:

(a) O ponto de interseção P se localiza no interior de T_1 e na aresta de F_1, logo prossegue-se para o teste de interseção entre T_1 e F_2; (b) O ponto de interseção P se localiza no interior de F_1 e na aresta de T_1, logo prossegue-se para o teste de interseção entre F_1 e T_2; (c) O Ponto de Interseção P se localiza no interior de F_1 e no vértice de T1, logo prossegue-se para os testes de interseção de F_1xT_2, F_1xT_3, F_1xT_4, F_1xT_5, das quais apenas F_1xT_2 realmente há interseção.

Fonte: LO & WANG(2004)

De fato, curvas de interseção são contínuas, logo, navegar pelas malhas em interseção por adjacências (triângulos e arestas vizinhas) é bastante natural. Isto requer, contudo, uma estrutura de dados para representar topologias de malhas de triângulos, como Radial-Edge (WEILER, 1988), Winged-Edge (BAUMGART, 1972) ou Half-Edge (MÄNTYLÄ, 1988). LO & WANG (2004) também chegaram a criar uma estrutura bem simples para se caminhar por adjacências em malhas.

Fazendo isso, diminui-se muito o número de pares triângulo-triângulo cujas interseções se precisa verificar. Na prática, um triângulo provavelmente só intersectará triângulos vizinhos a um triângulo que ele já intersectou. Assim, testa-se a interseção somente de um número reduzido de triângulos contemplados por buscas locais na malha. Entretanto, estas buscas exigem pares de triângulos iniciais para a construção de curvas de interseção. Para isso, o *background grid filter*, além de minimizar verificações de interseção de pares triângulo-triângulo, auxilia na escolha desses pares iniciais.

Além disso, elimina-se a necessidade de reordenarmos os pontos de interseção calculados, uma vez que, caminhado por adjacências, podemos criar curvas já ordenadas. Sendo assim, não convém utilizarmos interseções triângulos-triângulos, onde teremos pontos duplicados e gastaremos o dobro do esforço computacional. É mais prudente, nesse caso, utilizarmos interseções segmento-triângulo. A robustez destas interseções não é mais uma exigência do algoritmo de ordenação, mas é necessário para a correta determinação dos pontos das curvas de interseção.

3.2. Interseção Segmento-Triângulo

3.2.1. Descrição do Problema

Um segmento de reta *e*, representando uma aresta da malha, e um triângulo τ, representando um triângulo da outra malha, podem ser respectivamente definidos como:

$$e = \{X \in \mathcal{R}^3 | X = R + t(S - R)\} \quad (9)$$

para

$$\forall t \in \mathcal{R} | 0 \leq t \leq 1 \quad (10)$$

e

$$\tau = \{X \in \mathcal{R}^3 | X = A + u(B - A) + v(C - A)\} \quad (11)$$

para

$$\begin{cases} \forall u \in \mathcal{R} | 0 \leq u \leq 1 \\ \forall v \in \mathcal{R} | 0 \leq v \leq 1 \\ 0 \leq u + v \leq 1 \end{cases} \quad (12)$$

onde R e S são pontos extremos do segmento de reta *e* e A,B e C são vértices do triângulo τ.

Por sua vez, o plano deste triângulo, que será denominado de π, pode ser definido como:

$$\pi = \{X \in \mathcal{R}^3 | X = A + u(B - A) + v(C - A)\} \quad (13)$$

para *u* e *v* livres, isto é,

$$\begin{cases} \forall u \in \mathcal{R} \\ \forall v \in \mathcal{R} \end{cases} \quad (14)$$

Alternativamente, este plano pode ser definido por:

$$\pi = \{X \in \mathcal{R}^3 | X \cdot n_\pi + d_\pi = 0\} \quad (15)$$

para

$$n_\pi = (B - A) \times (C - A) \quad (16)$$

$$d_\pi = -A \cdot n_\pi \quad (17)$$

É importante ressaltar que a normal do triângulo τ, n_τ, é idêntica a normal do plano π, n_π. O triângulo τ, o plano π e o segmento *e* são exibidos na Figura 18.

Sendo assim, um ponto de interseção P entre o segmento *e* e o triângulo τ obedecerá, além das condições para *t*,*u* e *v* (Equações (10) e (12)), as seguintes equações:

$$P = R + t_P(S - R) \quad (18)$$

$$P = A + u_P(B - A) + v_P(C - A) \quad (19)$$

$$P \cdot n_\pi + d_\pi = 0 \quad (20)$$

Este ponto de interseção, P, também está sendo mostrado na Figura 18.

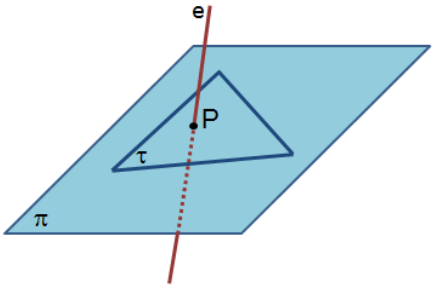


Figura 18: Triângulo τ, Plano π, Segmento *e* e Ponto P

Logo, haverá interseção somente se as três equações acima puderem ser simultaneamente respeitadas sem violar as condições para *t*, *u* e *v*. Caso não haja uma tripla de números reais *t*, *u* e *v* capazes de satisfazer essas três equações ou esta tripla existir mas desrespeitar as Equações (10) e (12), não haverá interseção entre *e* e τ.

A rigor, satisfazer a Equação (19) automaticamente soluciona a Equação (20), já que todo ponto pertencente ao triângulo τ pertencerá ao plano do triângulo π. A recíproca, entretanto, é falsa, já que nem todo ponto do plano do triângulo π pertencerá ao triângulo τ. Isto pode ser observado comparando as Equações (12) e (14). De fato, note, na Figura 19, que um ponto Q pode pertencer ao plano π estando do lado de fora das bordas de τ.

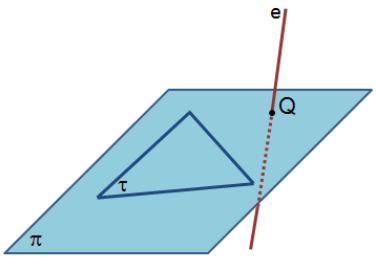
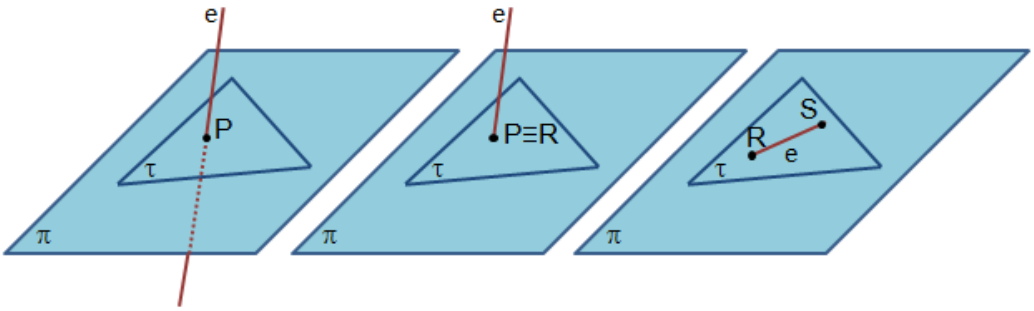


Figura 19: Ponto Q intersecta o plano π, mas não o triângulo τ.

Naturalmente, segmentos que sequer intersectam o plano π, não intersectam o triângulo τ. Tampouco segmentos paralelos e não-coplanares ao plano π. Só haverá interseção com o triângulo se o segmento, pelo menos, intersectar o plano π. Utilizando o plano do

triângulo para avaliação prévia da interseção, o ponto de interseção pode estar no plano do triângulo, mas fora das bordas do triângulo (como visto na Figura 19). A Figura 20 mostra os três casos onde o segmento *e* efetivamente intersecta o triângulo τ : segmento corta triângulo, segmento toca triângulo e segmento incluído no triângulo (completamente ou parcialmente).



(a) **(b)** **(c)**

Figura 20: Casos de Interseção Segmento-Triângulo

(a) Segmento corta Triângulo (b) Segmento toca Triângulo

(c) Segmento Incluído no Triângulo

Em todos esses casos, é preciso avaliar se o ponto de interseção realmente está dentro das bordas do triângulo. O caso onde o segmento é coplanar ao plano, por sua vez, pode ser decomposto em uma série de sub-casos. Em particular, o segmento pode entrar e sair do triângulo, somente entrar ou estar completamente no interior do triângulo.

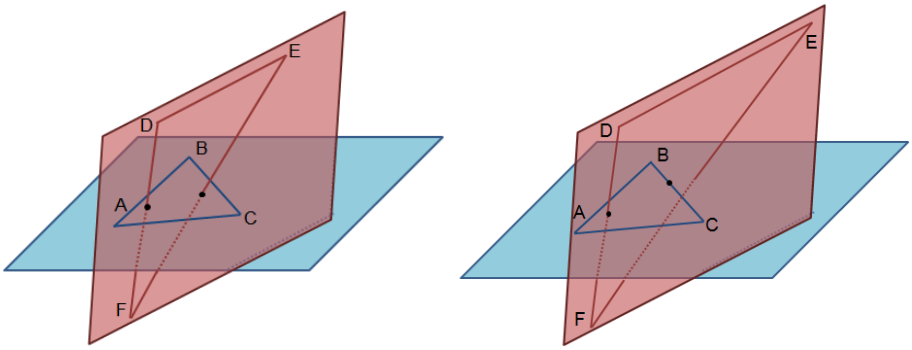
Os casos coplanares não serão tratados nesta dissertação, uma vez que a ferramenta proposta tem por finalidade gerar curvas de interseção que separem retalhos de superfícies. Arestas coplanares a triângulos é um caso típico de “superfícies em contato”, o que, devido ao tipo de modelagem realizada, só ocorre nas bordas de superfícies. A “colagem” de bordas de superfícies sobre outras superfícies foge ao tema desta dissertação.

3.2.2. Algoritmo de LO

LO (1995) propôs um algoritmo de interseção triângulo-triângulo baseado em 6 cálculos de interseção segmento-triângulo. Especificamente, ao se intersectar o triângulo ABC com o triângulo DEF, as seguintes interseções são verificadas:

- Segmento AB com triângulo DEF
- Segmento BC com triângulo DEF
- Segmento CA com triângulo DEF
- Segmento DE com triângulo ABC
- Segmento EF com triângulo ABC
- Segmento FD com triângulo ABC

A ideia de que todos os segmentos de um triângulo devem ter sua interseção verificada com o outro triângulo é bastante natural. Porém, a necessidade de se fazer o mesmo com as arestas do outro triângulo, ou seja, trocar-se o triângulo de referência, não é muito evidente. Para explicitar isso, a Figura 21 mostra duas possibilidades de interseção: Na primeira, dois segmentos de um mesmo triângulo cortam o outro triângulo. Na outra possibilidade, um segmento do primeiro triângulo corta o interior do segundo triângulo ao mesmo tempo em que um segmento do segundo triângulo corta o interior do primeiro triângulo. Assim, se não houver a troca do triângulo de referência, o segundo ponto de interseção não será calculado, o que justifica a necessidade desses seis testes de interseção.



(a) **(b)**

Figura 21: Possibilidades de Interseção com o Algoritmo de LO (1995)

(a) Segmento DF intersecta triângulo ABC e Segmento EF intersecta triângulo ABC

(b) Segmento DF intersecta triângulo ABC e Segmento BC intersecta triângulo DEF

A fundo, LO (1995), então, realiza interseções segmento-triângulo. Como vimos, um segmento somente intersectará um triângulo se, pelo menos, intersectar o plano desse triângulo, o que é uma condição necessária, porém não suficiente. Ainda assim, esta verificação é útil para descarte cedo de pares segmento-triângulo sem interseção. Havendo interseção, pode-se testar, em seguida, se o provável ponto de interseção está realmente dentro das bordas do triângulo.

Considerando o triângulo ABC e um segmento genérico RS (idênticos ao triângulo τ e ao segmento *e* da subseção anterior), haverá interseção em um único ponto (intermediário a R e S) somente se R e S estiverem em lados opostos do plano π. Isto pode ser expresso por:

$$\big((R - A) \cdot n_\pi\big) \cdot \big((S - A) \cdot n_\pi\big) < 0 \tag{21}$$

Caso o produto acima seja positivo, os pontos R e S estão do mesmo lado do plano π e, portanto, não há interseção. Caso o ponto R pertença ao plano π, o primeiro termo do produto acima será zero e caso o ponto S pertença ao plano π, o segundo termo deste produto será zero. Lembre-se que também há o caso onde ambos os pontos R e S pertencem a π.

Para se calcular o ponto de interseção intermediário P, cujo valor paramétrico no segmento *e* é dado por t_P, montamos o seguinte sistema:

$$\begin{cases} P = R + t_P(S - R) \\ P \cdot n_\pi + d_\pi = 0 \end{cases} \tag{22}$$

Resolvendo-o:

$$\big(R + t_P(S - R)\big) \cdot n_\pi + d_\pi = 0 \therefore$$
$$R \cdot n_\pi + t_P(S - R) \cdot n_\pi + d_\pi = 0 \therefore$$
$$t_P(S - R) \cdot n_\pi = -R \cdot n_\pi - d_\pi \therefore$$
$$t_P = -\frac{R \cdot n_\pi + d_\pi}{(S - R) \cdot n_\pi} \therefore$$

E, finalmente,

$$\boxed{t_P = \frac{R \cdot n_\pi + d_\pi}{(R - S) \cdot n_\pi}} \tag{23}$$

Em particular, LO (1995) calcula a seguinte fórmula, equivalente a acima:

$$\boxed{t_P = \frac{(R - A) \cdot n_\pi}{(R - A) \cdot n_\pi - (S - A) \cdot n_\pi}}$$

Note que, se R pertence ao plano, então o numerador de t_P será 0, t_P será 0 e P será igual a R. Se S pertence ao plano, então o numerador de t_P será igual ao denominador, então, t_P será 1 e P será igual a S. Se R e S pertencerem ao plano, então, o numerador e o denominador de t_P serão 0. Se t_P pertence ao intervalo [0,1], então há um ponto de interseção entre R e S. Caso contrário (se t_P estiver fora do intervalo [0,1]), então, o segmento e não intersecta o plano, embora uma reta na direção RS intersecte. De fato, se t_P for negativo ou maior que 1, o segmento estará estritamente acima ou estritamente abaixo do plano.

De qualquer forma, esses casos já foram tratados previamente pela condição da Equação (21). Assim, ao se calcular t_P, já caímos no caso de interseção onde há apenas um ponto de interseção, P, entre R e S. Então, calculamos o ponto P com a seguinte equação:

$$P = R + t_P(S - R) \tag{24}$$

e testa-se ele está dentro das bordas do triângulo como abaixo:

$$\begin{cases} (B - A) \times (P - A) \cdot n_\pi \geq 0 \\ (C - B) \times (P - B) \cdot n_\pi \geq 0 \\ (A - C) \times (P - C) \cdot n_\pi \geq 0 \end{cases} \tag{25}$$

Este algoritmo, porém, é muito suscetível a problemas numéricos. Para atingir as condições mencionadas nas Equações (21) e (25), os cálculos devem ser muito precisos. Se, na Equação (21), tivermos um produto de número positivo com um número positivo bem baixo que deveria ser negativo, estaremos incorretamente descartando um ponto de interseção. Idem se tivermos um número negativo sendo multiplicado com um número negativo bem baixo que deveria ser positivo.

Da mesma forma, se tivermos um número positivo e um número negativo bem baixo que deveria ser positivo, então, estaremos incluindo um ponto de interseção bem perto das extremidades do segmento e que não deveria existir. Idem se tivermos um número negativo e um número positivo bem baixo que deveria ser negativo.

Outro problema é que dificilmente os valores da Equação (21) serão exatamente zero na situação em que deveriam, devido a erros numéricos, geralmente, atingindo valores positivos ou negativos bem pequenos. Na Equação (21), se tivermos um valor negativo e um positivo bem baixo, teremos um ponto de interseção intermediário bem próximo a uma das extremidades do segmento, porém, ele não será propriamente a extremidade. Idem para um valor positivo e um valor negativo bem baixo.

Mais difícil ainda seria obtermos dois valores exatamente zero, como previsto na condição da Equação (21) no caso em que R e S pertencem ao plano. Caso tenhamos dois valores positivos bem baixos ou dois valores negativos bem baixos, o segmento será erroneamente classificado como sem interseção com o triângulo. Mas, caso haja um valor positivo bem baixo e um valor negativo bem baixo, então, calcularemos um ponto de interseção P cujo posicionamento será dado de acordo com os erros numéricos cometidos. Ou seja, sua localização será imprevisível. Caso não haja verificação de se t_P pertence ao intervalo [0,1], este ponto de interseção poderá inclusive, localizar-se fora do segmento RS na direção da reta RS.

Fora isso, a imprecisão no cálculo do ponto intermediário P e em sua avaliação na Equação (25), podem levar ao incorreto descarte deste ponto de interseção. Outra possibilidade seria a incorreta inclusão de um ponto fora do triângulo. Assim, usar este algoritmo não é seguro.

Podemos, contudo, utilizar os Predicados de Shewchuk para descartar, com segurança, pares segmento-triângulo que não estão em interseção. Ao mesmo tempo, o uso de aritmética exata nos garante maior precisão no cálculo ponto intermediário.

3.2.3. Algoritmo de MOLLER & TRUMBORE

Um algoritmo bem mais econômico foi criado por MÖLLER & TRUMBORE (1997). Partindo da equação paramétrica de um segmento e da equação baricêntrica de um triângulo, temos o seguinte sistema:

$$\begin{cases} P = R + t_P(S - R) \\ P = A + u_P(B - A) + v_P(C - A) \end{cases} \tag{26}$$

E convertendo-o para a forma matricial, temos:

$$R + t_P(S - R) = A + u_P(B - A) + v_P(C - A)$$
$$t_P(R - S) + u_P(B - A) + v_P(C - A) = R - A$$
$$\begin{bmatrix} R_x - S_x & B_x - A_x & C_x - A_x \\ R_y - S_y & B_y - A_y & C_y - A_y \\ R_z - S_z & B_z - A_z & C_z - A_z \end{bmatrix} \begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \begin{bmatrix} R_x - A_x \\ R_y - A_y \\ R_z - A_z \end{bmatrix} \tag{27}$$

Resolvendo-o pela Regra de Cramer, teremos o seguinte denominador, comum a todas as variáveis:

$$\Delta = \begin{vmatrix} R_x - S_x & B_x - A_x & C_x - A_x \\ R_y - S_y & B_y - A_y & C_y - A_y \\ R_z - S_z & B_z - A_z & C_z - A_z \end{vmatrix} = \begin{vmatrix} R_x - S_x & R_y - S_y & R_z - S_z \\ B_x - A_x & B_y - A_y & B_z - A_z \\ C_x - A_x & C_y - A_y & C_z - A_z \end{vmatrix}$$
$$\Delta = (R - S) \cdot (B - A) \times (C - A) \tag{28}$$

O numerador relativo a t_P será:

$$\Delta t_P = \begin{vmatrix} R_x - A_x & B_x - A_x & C_x - A_x \\ R_y - A_y & B_y - A_y & C_y - A_y \\ R_z - A_z & B_z - A_z & C_z - A_z \end{vmatrix} = \begin{vmatrix} R_x - A_x & R_y - A_y & R_z - A_z \\ B_x - A_x & B_y - A_y & B_z - A_z \\ C_x - A_x & C_y - A_y & C_z - A_z \end{vmatrix}$$

$$\Delta t_P = (R - A) \cdot (B - A) \times (C - A) \tag{29}$$

Já o numerador relativo a u_P será:

$$\Delta u_P = \begin{vmatrix} R_x - S_x & R_x - A_x & C_x - A_x \\ R_y - S_y & R_y - A_y & C_y - A_y \\ R_z - S_z & R_z - A_z & C_z - A_z \end{vmatrix} = \begin{vmatrix} R_x - S_x & R_y - S_y & R_z - S_z \\ R_x - A_x & R_y - A_y & R_z - A_z \\ C_x - A_x & C_y - A_y & C_z - A_z \end{vmatrix}$$

$$\Delta u_P = (R - S) \cdot (R - A) \times (C - A) \tag{30}$$

E o último numerador, relativo a v_P, será:

$$\Delta v_P = \begin{vmatrix} R_x - S_x & B_x - A_x & R_x - A_x \\ R_y - S_y & B_y - A_y & R_y - A_y \\ R_z - S_z & B_z - A_z & R_z - A_z \end{vmatrix} = \begin{vmatrix} R_x - S_x & R_y - S_y & R_z - S_z \\ B_x - A_x & B_y - A_y & B_z - A_z \\ R_x - A_x & R_y - A_y & R_z - A_z \end{vmatrix}$$

$$\Delta v_P = (R - S) \cdot (B - A) \times (R - A) \tag{31}$$

Finalmente, a solução do sistema será:

$$\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{\Delta} \begin{bmatrix} \Delta t_P \\ \Delta u_P \\ \Delta v_P \end{bmatrix}$$

$$\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{(R - S) \cdot (B - A) \times (C - A)} \begin{bmatrix} (R - A) \cdot (B - A) \times (C - A) \\ (R - S) \cdot (R - A) \times (C - A) \\ (R - S) \cdot (B - A) \times (R - A) \end{bmatrix} \tag{32}$$

MÖLLER & TROMBORE (1997) fizeram otimizações para reduzir o número de operações aritméticas realizadas. Primeiro, eles reescreveram a solução do sistema como:

$$\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{(B-A)\cdot(S-R)\times(C-A)} \begin{bmatrix} (C-A)\cdot(R-A)\times(B-A) \\ (R-A)\cdot(S-R)\times(C-A) \\ (S-R)\cdot(R-A)\times(B-A) \end{bmatrix} \tag{33}$$

Depois, eles armazenam os vetores diferença em variáveis

$$D = S - R$$
$$E_1 = B - A$$
$$E_2 = C - A$$
$$F = R - A$$

e também acumulam os resultados dos seguintes produtos vetoriais

$$K = D \times E_2 = (S-R)\times(C-A)$$
$$L = F \times E_1 = (R-A)\times(B-A)$$

Assim, eles reaproveitam cálculos e a solução final do sistema pode ser obtida de forma bem simples:

$$\boxed{\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{E_1 \cdot K} \begin{bmatrix} E_2 \cdot L \\ F \cdot K \\ D \cdot L \end{bmatrix}} \tag{34}$$

4

Algoritmo Proposto

4.1. Seleção de Candidatos a Interseção

O algoritmo que essa dissertação propõe começa determinando quais pares de triângulos estão provavelmente em interseção. Em outras palavras, será feita uma seleção de pares de triângulos candidatos a estarem em interseção. A estratégia adotada supõe que as superfícies originais não possuem auto-interseções e, portanto, cada um estes pares candidatos deve apresentar dois triângulos, um de cada malha.

Dois triângulos quaisquer só podem estar em interseção se estiverem espacialmente próximos. Por esse motivo, contaremos com auxílio de uma estrutura de subdivisão espacial hierárquica chamada R-Tree (GUTTMAN 1984). A Figura 22 mostra o exemplo de uma R-Tree de um conjunto de pontos 3D. Cada nó da R-Tree representa uma célula (paralelepípedo em 3D ou um retângulo em 2D) contendo outras células. Uma folha da R-Tree representa uma célula contendo somente um determinado número de pontos, que não estão mais distribuídos em outras subcélulas.

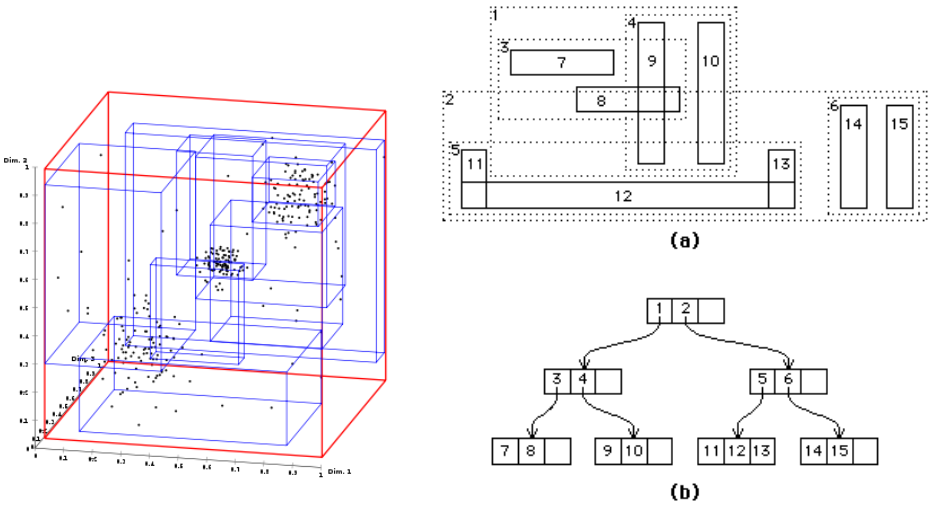


Figura 22: Exemplo de R-Tree de um conjunto de pontos 3D.

Para determinarmos os pares de candidatos, primeiro, constrói-se uma R-Tree para os triângulos da primeira malha. Assim, as folhas dessa R-Tree contém apenas Axis-Aligned *Bounding Boxes* (caixas envolventes alinhadas aos eixos canônicos) de cada triângulo dessa malha. Em seguida, os triângulos da segunda malha são adicionados a essa R-Tree, verificando em qual célula-folha da R-Tree original, esses novos triângulos estariam contidos. Havendo colisão dentro de uma dessas células-folha, este par de triângulos será um candidato a par de triângulos em interseção.

Se um triângulo da segunda malha interceptar mais de uma célula-folha, então, vários pares de candidatos deverão ser criados, envolvendo justamente esse triângulo com o triângulo original das células que ele cortou. Note que um triângulo da primeira malha pode formar um par candidato a interseção com mais de um triângulo da segunda malha e vice-versa.

Como ainda não se está utilizando aritmética exata, essas caixas envolventes são expandidas em 2% de seu tamanho original antes dos testes de colisão, assim evitando que erros numéricos deturpem o resultado da verificação. Com este artifício, impede-se que pares de triângulos sejam erroneamente descartados como candidatos. Porém, ao mesmo tempo, pares de triângulos sem estar em interseção são mais facilmente aceitos como candidatos. Na prática, mesmo sem este artifício, dois triângulos que pertençam à mesma célula podem não possuir interseção, logo um teste mais rigoroso seria necessário de qualquer forma nas etapas futuras do algoritmo.

Se um par de candidatos não tiver realmente interseção, ao se tentar utilizar este par para se iniciar a construção de uma curva de interseção, os testes de interseção segmento-triângulo realizados com aritmética exata mostrarão que este par não possui interseção e não é possível construir uma curva de interseção a partir dele.

4.2. Busca por Interseções

Por sua vez, os pontos de interseção serão incluídos nas curvas de interseção conforme uma busca realizada em ambas as malhas, partindo

desses pares de candidatos a triângulos em interseção. Esta navegação pelas malhas conta com auxílio de uma estrutura de dados topológica bem simples para se representar triangulações, chamada *cTopology.* Entretanto, qualquer estrutura de dados semelhante pode ser usada. Esta estrutura de dados contém as seguintes informações:

- Coordenadas X, Y e Z de cada nó;
- *Flag* (de uso livre) para cada nó, aresta e triângulo;
- Noção de nós e arestas da borda (fronteira) da superfície;
- Nós e arestas de cada triângulo;
- Nós adjacentes a nós, arestas adjacentes a nós e triângulos adjacentes a nós;
- Nós adjacentes a arestas e triângulos adjacentes a arestas;
- Arestas adjacentes a triângulos.

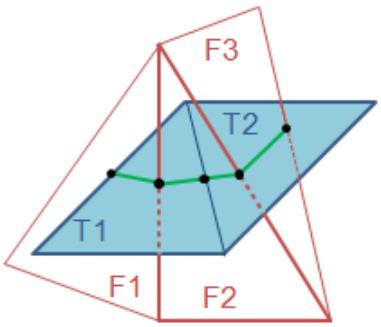
A busca por pares triângulo-triângulo em interseção segue o estilo de uma busca em profundidade (*Depth-First Search*) em um grafo bidirecional. Dado um par de triângulos, T_A e T_B, para cada aresta e_A não-visitada de T_A, havendo interseção entre e_A e T_B, adiciona-se o(s) ponto(s) de interseção na curva e, recursivamente, procura-se por mais interseções nos seguintes pares de triângulo:

- T_A e T_B
- Adj(e_A) e T_B
- T_A e Adj(T_B)
- Adj(T_A) e Adj(T_B)

onde Adj(k) representa os triângulos adjacentes a aresta/triângulo k. Em seguida, deve-se fazer o contrário. Para cada aresta e_B não-visitada de T_B, havendo interseção entre e_B e T_A, adiciona-se o(s) ponto(s) de interseção na curva e, recursivamente, procura-se por mais interseções nos seguintes pares de triângulo:

- T_B e T_A
- Adj(e_B) e T_A
- T_B e Adj(T_A)
- Adj(T_B) e Adj(T_A)

Apesar da grande abrangência da busca realizada, o algoritmo não “explode” porque cada aresta só é visitada uma vez. Na prática, nossa busca avança como na Figura 23. Na Figura 23(a) pode-se visualizar a curva sendo construída e, na Figura 23(b), o passo-a-passo dos testes de interseção realizados.



(a)

F1 x T1 → ✓
| F1 x T1 → ✓
| | F1 x T1 → ×
| | F2 x T1 → ✓
| | | F2 x T1 → ✓
| | | | F2 x T1 → ×
| | | | F3 x T1 → ×
| | | | F2 x T2 → × (aresta já visitada)
| | | | F3 x T2 → ✓
| | | | | F3 x T2 → ✓
| | | | | | F3 x T2 → ×
| | | | | | Aresta de F3 sem adj → ×
| | | | | | F3 x T1 → ×
| | | | | | F2 x T1 → × (já visitada)
| | | | | [...]→×
| | | | [...]→×
| | | [...]→×
| | [...]→×
| [...]→×
[...]→×

(b)

Figura 23: Avanço da Busca por Interseções

(a) Visualização da busca realizada; (b) Passo-a-passo da busca realizada

Se F1 teve interseção com T1, continua-se verificando outra interseção nesse mesmo par. Na figura, houve outra interseção no interior de T1 com uma aresta de F1. Como não há mais nenhuma aresta não-visitada de F1 em interseção com T1, a busca avança para o triângulo F2, adjacente ao da aresta intersectada, e mantém-se no triângulo T1. A aresta que F1 compartilha com F2 já foi visitada e, portanto, a interseção de F2 com T1 será em um novo ponto na aresta compartilhada de T1 com T2. Neste momento, todas as arestas de F1 já foram visitadas e segue-se para o triângulo adjacente a F1 pela aresta em que foi intersectada pela última vez. Logo, segue-se testando a interseção de T2 com F2 e assim por diante.

O método TNOIT, de LO & WANG (2004), é semelhante a busca proposta acima. Uma diferença é que, dependendo do caso de interseção Segmento-Triângulo (ponto de interseção no vértice, aresta ou interior do triângulo), LO & WANG (2004) partem para um par de entidades vizinhas em interseção específico. Já no algoritmo proposto realiza-se uma busca mais genérica.

Repare, por exemplo, que verificamos até se triângulos imediatamente vizinhos aos intersectados também se intersectam. Isto foi necessário porque o TNOIT lida com malhas regulares, onde o cruzamento de triângulos é mais bem comportado. Assim, é assumido que, após um triângulo T1 intersectar o meio do triângulo F1, o triângulo vizinho a T1, T2, intersectará o triângulo F1 novamente, o que não é verdade para malhas irregulares. Os testes realizados mostraram que, sem verificar se triângulos vizinhos se intersectam, as curvas de interseção ficaram interrompidas.

Na Figura 24, é possível observar que a aresta de T1 intersecta o interior de F1. Nesse caso, como T1 foi intersectado em uma de suas arestas, o TNOIT prosseguirá tentando intersectar T2 com F1. A aresta de T2 que intersecta F1, porém, já foi visitada (através do triângulo F1). Se o método TNOIT permite re-visitar arestas, haverá pontos duplicados na curva de interseção, o que é indesejado. Por outro lado, caso não permita, não haverá uma nova interseção e o método TNOIT irá terminar, sem concluir a curva de interseção, o que é uma falha grave.

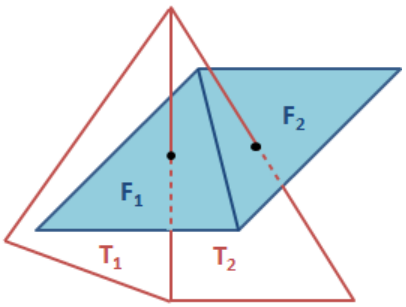


Figura 24: Malha irregular onde TNOIT não funciona bem

Assim, esta dissertação propõe um novo método de busca de interseções, cujo pseudocódigo está sendo mostrado na Figura 25. Tal método, denominado por *Search_Intersections,* chama a função *Segment_Triangle_Intersection* que será explicada em mais detalhes na próxima subseção. Por hora, basta saber que a função *Segment_Triangle_Intersection* calcula a interseção entre um segmento (ou aresta) e um triângulo, retornando "*true"* caso haja interseção ou "*false"* caso contrário.

```
Search_Intersections(TA, TB):
For each unvisited edge eA of triangle TA
   Mark eA as visited
   If Segment_Triangle_Intersection(eA,TB) then
    Search for more intersections between TA and TB
    Search for more intersections between the triangles adjacent to eA and TB
    Search for more intersections between TA and the triangles adjacent to TB
    Search for more intersections between the triangles adjacent to TA and the triangles adjacent to TB
     End If
End For
For each unvisited edge eB of triangle TB
   Mark eB as visited
   If Segment_Triangle_Intersection(eB,TA) then
    Search for more intersections between TB and TA
    Search for more intersections between the triangles adjacent to eB and TA
    Search for more intersections between TB and the triangles adjacent to TA
    Search for more intersections between the triangles adjacent to TB and the triangles adjacent to TA
     End If
End For
```

Figura 25: Algoritmo de Busca por Pares Triângulo-Triângulo em Interseção (Método Search_Intersections)

O método *Search_Intersections* constrói, recursivamente, uma curva do tipo *chain* a partir de um par de triângulos inicial em interseção. Se fornecermos um par de triângulos inicial sem interseção, apenas as arestas desses triângulos serão marcadas como visitadas. Contudo, se o par de triângulos inicial estiver em interseção, mas for do meio da *chain*, a curva irá do meio até uma das pontas, retornará abruptamente ao meio e seguirá para a outra ponta.

Para evitarmos isto, os pares fornecidos inicialmente a esta função deverão possuir pelo menos um triângulo de borda. Um triângulo é considerado de borda se pelo menos uma de suas arestas é da borda da superfície. De posse de uma lista de pares de triângulos em interseção, pegaremos apenas os da borda, o que é uma consulta simples a nossa estrutura de dados topológica da malha. De preferência, começamos chamando esta função pelos pares onde ambos os triângulos sejam da borda.

Assim, cada vez que chamarmos esta função, estaremos criando um *chain*, indo da borda de uma das superfícies até a borda de uma das superfícies (que não precisa necessariamente ser a mesma).

É importante ressaltar que, caso a superfície possua buracos (vários *genera*), a *cTopology* considera as arestas da borda dos buracos como da borda da superfície. Assim, naturalmente se produz *chains* da borda da superfície até um buraco, de um buraco a outro buraco, e de buracos para borda da superfície.

Caso não haja mais pares não-visitados onde ambos os triângulos sejam da borda, prosseguimos chamando esta função pelos pares contendo apenas um triângulo de borda. Após todos os pares desse tipo terem sido visitados, se ainda sobrar pares de triângulos a serem visitados, eles certamente pertencerão a curvas do tipo *loop*.

A rigor, uma curva do tipo *loop* não possui pontos inicial nem final. Portanto, poderíamos disparar o método *Search_Intersections* de qualquer par ainda não-visitado. Então, a curva não teria problema de retornar abruptamente para seu início, porém, o último ponto da curva de interseção não estaria ligado ao primeiro, uma vez que pertenceria a um par já visitado. Como solução, ao terminarmos nossa busca por

interseções em uma curva do tipo *loop*, basta repetirmos o primeiro ponto. Fazendo uma simples cópia, nosso esforço computacional é mínimo.

Assim, nosso algoritmo para a construção de todas as curvas de interseção, independente do número de loops e chains a serem construídos, está sendo exibido na Figura 26.

```
Build_All_Intersection_Curves(M_A, M_B):
Using an R-Tree, build a list of candidates for triangle-triangle intersection pairs between meshes M_A and M_B
For each unvisited pair of triangles (T_A ,T_B) where both triangles are boundary triangles
    Start building a new intersection curve
    Search_Intersections (T_A, T_B)
    End building current intersection curve
End For
For each unvisited pair of triangles (T_A ,T_B) containing a boundary triangle
    Start building a new intersection curve
    Search_Intersections (T_A, T_B)
    End building current intersection curve
End For
For each unvisited pair of triangles (T_A ,T_B)
    Start building a new intersection curve
    Search_Intersections (T_A, T_B)
    Include a last intersection point, identical to the first one.
    End building current intersection curve
End For
```

Figura 26: Algoritmo para Construir Todas as Curvas de Interseção (Método Build_All_intersection_Curves)

4.3. Interseção Segmento-Triângulo

No algoritmo proposto, se ambos os pontos extremos do segmento em questão estiverem estritamente acima ou estritamente abaixo do plano do triângulo, não haverá interseção. Além disso, os Predicados de Shewchuk podem nos dizer qual dos três casos de interseção com o plano do triângulo está ocorrendo. A Tabela 1 mostra a correlação entre os sinais dos predicados e os casos de interseção do segmento RS com o plano de um triângulo ABC, orientado no sentido anti-horário.

Note que, se o triângulo ABC estiver orientado no sentido horário, os sinais da Tabela 1 irão se inverter, porém, o candidato a ponto de interseção será o mesmo. A fundo, a única diferença é que um descarte por “segmento acima do plano” passa a ser um descarte por “segmento abaixo do plano”.

Classificação	Sinal de *orient3d*(A,B,C,R)	Sinal de *orient3(A,B,C,S)*	Candidato a Ponto de Interseção
Segmento Abaixo do Plano	+	+	Nenhum
Segmento toca Plano, por baixo	0	+	R
	+	0	S
Contido no Plano	0	0	R e S
Segmento toca Plano, por cima	0	-	R
	-	0	S
Segmento Acima do Plano	-	-	Nenhum
Segmento corta Plano	+	-	Entre R e S
	-	+	Entre R e S

Tabela 1: Classificação dos Casos de Interseção do Segmento com o Plano do Triângulo segundo o Algoritmo Proposto

Mesmo havendo interseção com o plano do triângulo, porém, pode não estar ocorrendo interseção com o triângulo (vide Figura 19). Um ponto extremo ou intermediário do segmento que pertence ao plano do triângulo será ponto de interseção apenas se estiver dentro das bordas do triângulo. Lembre-se que, se ambos os pontos extremos do segmento forem simultaneamente coplanares, consideraremos que não há interseção.

Se já detectarmos que apenas os pontos extremos do segmento são reais candidatos a ponto de interseção, basta verificarmos se eles estão dentro das bordas do triângulo, o que explicaremos na seção 4.3.2. Senão, precisaremos calcular o ponto de interseção, nesse caso, um ponto intermediário do segmento.

A maneira de calcularmos este ponto intermediário e avaliar se ele está dentro das bordas do triângulo será mostrada a partir do Algoritmo de Moller & Trumbore (1997). Entretanto, será preciso introduzir aritmética

exata neste algoritmo, o que será feito utilizando os predicados *orient2d* e *orient3d*. Em outras palavras, calcularemos a solução do sistema de equações de Moller & Trumbore (1997) de forma exata.

4.3.1. Solução Exata do Sistema de Equações

O denominador obtido por Moller & Trumbore (1997) foi:

$$\Delta = (R - S) \cdot (B - A) \times (C - A)$$

Para os predicados de Shewchuk, entretanto, precisamos de uma origem comum às subtrações vetoriais do produto misto, de forma a termos um tetraedro (4 vértices). Ou seja, precisamos de um produto misto da forma:

$$(U - O) \cdot (V - O) \times (W - O)$$

Infelizmente, nosso denominador não se encontra escrito desta maneira. Porém, podemos convertê-lo para uma subtração dois produtos mistos:

$$\Delta = (R - A) \cdot (B - A) \times (C - A) - (S - A) \cdot (B - A) \times (C - A)$$

Note que o minuendo desta subtração é igual a Δt_P na Equação (29). Por conveniência, chamaremos o subtraendo desta subtração de Δs_P . Assim:

$$\Delta t_P = (R - A) \cdot (B - A) \times (C - A)$$
$$\Delta s_P = (S - A) \cdot (B - A) \times (C - A)$$
$$\Delta = \Delta t_P - \Delta s_P$$

Quando Δt_P e Δs_P possuírem sinais iguais, o segmento RS não intersectará nem o plano do triangulo ABC. Portanto, não haverá interseção com o triângulo ABC. Contudo, no caso contrário, o ponto de

interseção poderá pertencer ao plano do triângulo ABC, porém, do lado externo ao triângulo ABC. Assim, é necessário verificar se o ponto de interseção estará dentro das bordas do triângulo, o que será feito mais adiante. Por hora, voltemos ao cálculo do denominador.

Com a Equação (6), podemos calcular dois termos do denominador com aritmética exata:

$$\Delta = orient3d(A, B, C, R) - orient3d(A, B, C, S) \qquad (35)$$

Agora, para definirmos Δ com todos apenas efetuarmos uma subtração de números reais com aritmética exata. Para isso, sabendo-se que:

$$a - b = \begin{vmatrix} a - 0 & b - 0 \\ 1 - 0 & 1 - 0 \end{vmatrix}$$

podemos definir a subtração de escalares com aritmética exata como:

$$a \ominus b = orient2d((a, b), (1,1), (0,0)) \qquad (36)$$

Por convenção, os operadores aritméticos circulados denotarão, ao longo desta dissertação, operações com aritmética exata, vetoriais ou escalares, dependendo do tipo de seus operandos.

Assim, nosso denominador será exatamente:

$$\Delta = orient2d\left(\left(orient3d(A, B, C, R), orient3d(A, B, C, S)\right), (1,1), (0,0)\right) \qquad (37)$$

Ou, escrevendo-o de forma mais compacta:

$$\boxed{\Delta = orient3d(A, B, C, R) \ominus orient3d(A, B, C, S)} \qquad (38)$$

Portanto, o custo do denominador, na prática, é de um predicado *orient3d* relativo ao subtraendo e mais um predicado *orient2d* devido a subtração, já que o minuendo do denominador é um predicado previamente calculado no numerador de t_P.

Podemos, partindo do mesmo princípio, decompor Δu_P :

$$\Delta u_P = (R-S)\cdot(R-A)\times(C-A)$$
$$\Delta u_P = (R-A)\cdot(R-A)\times(C-A) - (S-A)\cdot(R-A)\times(C-A)$$

Como o primeiro produto misto possui dois termos iguais, ele certamente é zero. Geometricamente, o paralelepípedo definido por dois vetores iguais, além de um terceiro vetor qualquer, representa um paralelogramo 2D, ou seja, uma entidade de volume zero. Logo,

$$\Delta u_P = -(S-A)\cdot(R-A)\times(C-A)$$
$$\Delta u_P = (S-A)\cdot(C-A)\times(R-A)$$
$$\Delta u_P = orient3d(S,C,R,A)$$
$$\boxed{\Delta u_P = orient3d(A,R,C,S)} \qquad (39)$$

E fazendo algo semelhante para Δv_P:

$$\Delta v_P = (R-S)\cdot(B-A)\times(R-A)$$
$$\Delta v_P = \underbrace{(R-A)\cdot(B-A)\times(R-A)}_{0} - (S-A)\cdot(B-A)\times(R-A)$$
$$\Delta v_P = -(S-A)\cdot(B-A)\times(R-A)$$
$$\Delta v_P = (S-A)\cdot(R-A)\times(B-A)$$
$$\Delta v_P = orient3d(S,R,B,A)$$
$$\boxed{\Delta v_P = orient3d(A,B,R,S)} \qquad (40)$$

Portanto, nossa solução com os predicados fica:

$$\boxed{\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{orient3d(A,B,C,R) \ominus orient3d(A,B,C,S)} \begin{bmatrix} orient3d(A,B,C,R) \\ orient3d(A,R,C,S) \\ orient3d(A,B,R,S) \end{bmatrix}}$$

(41)

onde Θ significa a subtração com aritmética exata como visto na Eq (36)

Como estamos interessados em u_P e v_P apenas para descarte, podemos invertê-los, obtendo uma fórmula mais mnemônica:

$$\begin{bmatrix} t_P \\ u_P \\ v_P \end{bmatrix} = \frac{1}{orient3d(A,B,C,R) \ominus orient3d(A,B,C,S)} \begin{bmatrix} orient3d(A,B,C,R) \\ orient3d(A,B,R,S) \\ orient3d(A,R,C,S) \end{bmatrix}$$

(42)

Assim, conforme descemos pelas linhas dessa fórmula, o parâmetro R segue a diagonal secundária das listas de parâmetros. Repare que, com base nesta fórmula, podemos construir um algoritmo que utiliza apenas 4 predicados *orient3d* no caso médio (onde há apenas um ponto de interseção).

4.3.2. Condições de Pertinência ao Triângulo

Usaremos apenas u_P e v_P para descarte e calcularemos t_P somente quando necessário, para calcular um ponto de interseção diferente de R e S. Se R e S forem paralelos ao plano do triângulo ABC, os predicados *orient3d(A,B,C,S)* e *orient3d(A,B,C,R)* serão iguais. Nesta situação, os dois podem ser iguais a zero, o que significa que o segmento RS é coplanar ao triângulo ABC. Mas se os dois forem positivos ou negativos, o segmento RS estará respectivamente acima ou abaixo do plano e, portanto, não estará em interseção. Somente quando esses predicados tiverem sinais opostos, haverá um ponto de interseção intermediário ao segmento, demandando cálculo de t_P.

Assim, o valor de t_P, quando calculado, certamente estará no intervalo [0,1], por causa do nosso tratamento prévio, porém u_P e v_P não necessariamente. Se u_P e v_P valor estiverem fora do intervalo [0,1], não haverá interseção com o triângulo, logo, estes valores devem ter prioridade em nosso tratamento prévio. A rigor, temos que satisfazer:

$$\begin{cases} 0 \leq u_P \leq 1 \\ 0 \leq v_P \leq 1 \\ 0 \leq u_P + v_P \leq 1 \end{cases} \tag{43}$$

para

$$u_P = {\Delta u_P}/{\Delta}$$

$$v_P = {\Delta v_P}/{\Delta} \tag{44}$$

com $\Delta, \Delta u_P$ e Δv_P definidos respectivamente nas Equações (39),(40) e (38).

Entretanto, é possível realizar as comparações de u_P e v_P sem fazer divisões. Como veremos a seguir, será necessário apenas uma soma exata e operadores relacionais, como “maior que”(>), “maior ou igual a”(>=) e etc. Por conveniência, incluiremos também operadores lógicos, como conjunção(“e” / “*and*”) e disjunção(“ou” / “*or*”), evitando uma série maior de testes condicionais(“se-então-senão” / “*if-then-else*”). Em particular, utilizaremos também uma multiplicação convencional para simplificarmos expressões booleanas, o que é sabidamente mais eficiente.

Primeiro, determinaremos a expressão lógica que satisfaz:

$$0 \leq u_P \leq 1 \tag{45}$$

Sabe-se que o quociente de uma divisão entre números reais é negativo se, e somente se, seu dividendo e seu divisor possuem sinais opostos. Assim, u_P só será negativo se Δ for positivo e Δu_P for negativo ou se Δ for negativo e Δu_P for positivo. Formalmente,

$$u_p < 0 \Leftrightarrow \left((\Delta < 0) \wedge (\Delta u_p > 0)\right) \vee \left((\Delta > 0) \wedge (\Delta u_p < 0)\right)$$

Ou, de forma mais eficiente:

$$u_p < 0 \iff \Delta \cdot \Delta u_p < 0$$

(46)

Como estamos interessados apenas no sinal, este produto não precisa ser realizado com aritmética exata, logo, demandando baixo esforço computacional.

De forma semelhante, sabe-se que apenas frações impróprias correspondem a números maiores que uma unidade. Uma fração entre número reais é imprópria se, e somente se, seu denominador for menor que seu numerador. Assim, u_P será maior que 1 somente se seu numerador, Δu_P, for maior que seu denominador. Formalmente,

$$u_p > 1 \iff \Delta u_p > \Delta$$

(47)

Portanto, só haverá interseção se:

$$\sim\Big(\big(\Delta \cdot \Delta u_p < 0\big) \vee \big(\Delta < \Delta u_p\big)\Big)$$

Ou, de forma equivalente,

$$\Big(\sim\big(\Delta \cdot \Delta u_p < 0\big) \wedge \sim\big(\Delta < \Delta u_p\big)\Big)$$

Ou ainda

$$\Big(\big(\Delta \cdot \Delta u_p \geq 0\big) \wedge \big(\Delta \geq \Delta u_p\big)\Big)$$

(48)

Uma expressão análoga a esta satisfaz as condições para v_P:

$$\Big(\big(\Delta \cdot \Delta v_p \geq 0\big) \wedge \big(\Delta \geq \Delta v_p\big)\Big)$$

(49)

E há mais uma condição a ser verificada:

$$0 \leq u_p + v_p \leq 1$$

(50)

Após passarmos pelos dois testes anteriores, tanto u_P quanto v_P são números positivos, logo, não há chance da soma deles dar menor que zero. Assim, basta verificar que:

$$u_p + v_p \leq 1$$

(51)

Novamente, basta fazermos uma comparação entre o denominador e os numeradores:

$$\Delta v_p + \Delta u_p \leq \Delta$$

(52)

Esta soma pode ser expressa pela variável:

$$\Delta\overline{w_P} = \Delta u_P + \Delta v_P$$

(53)

onde a variável $\Delta\overline{w_P}$ representa o numerador do complemento da 3ª coordenada baricêntrica do triângulo. Explicaremos o significado desta variável partindo da Equação (12)

$$w_P = 1 - u_P - v_P$$

Logo o complemento de w_P. será:

$$\overline{w_P} = 1 - w_P$$

$$\overline{w_P} = 1 - \big(1 - (u_P + v_P)\big)$$

$$\overline{w_P} = u_P + v_P$$

E, portanto,

$$\frac{\overline{\Delta w_P}}{\Delta} = \frac{\Delta u_P}{\Delta} + \frac{\Delta v_P}{\Delta}$$

$$\Delta\overline{w_P} = \Delta u_P + \Delta v_P$$

Isso, entretanto, requer uma soma, que deveria ser feita com aritmética exata. Sabendo-se que:

$$a + b = \begin{vmatrix} a - 0 & b - 0 \\ -1 - 0 & 1 - 0 \end{vmatrix} \tag{54}$$

podemos definir a soma de escalares com aritmética exata como:

$$a \oplus b = orient2d((a, b), (-1,1), (0,0)) \tag{55}$$

Dessa forma, podemos definir a soma exata de Δu_P com Δv_P como:

$$\Delta\overline{w_P} = \Delta v_p \oplus \Delta u_p \tag{56}$$

Assim, basta verificarmos se

$$\Delta\overline{w_P} \leq \Delta \tag{57}$$

Unindo todas as condições que obtivemos até aqui, teremos:

$$\begin{cases} \left((\Delta \cdot \Delta u_p \geq 0) \wedge (\Delta \geq \Delta u_p)\right) \\ \left((\Delta \cdot \Delta v_p \geq 0) \wedge (\Delta \geq \Delta v_p)\right) \\ \Delta\overline{w_P} \leq \Delta \end{cases} \tag{58}$$

Porém, essas condições se aplicam apenas a triângulos de um lado. Como se trata de uma operação de modelagem, precisamos levar em conta os dois lados dos triângulos (isto é, sem *back-face culling*). Logo, precisaremos aceitar que Δu_P e Δv_P sejam negativos desde que Δ também o seja. Portanto, para haver um ponto de interseção dentro das bordas do triângulo, considerando ambos seus lados, deve-se verificar:

$$\begin{cases} \Big(\big(\Delta \cdot \Delta u_p \geq 0 \big) \wedge \big(|\Delta| \geq |\Delta u_p| \big) \Big) \\ \Big(\big(\Delta \cdot \Delta v_p \geq 0 \big) \wedge \big(|\Delta| \geq |\Delta v_p| \big) \Big) \\ |\Delta \overline{w_P}| \leq |\Delta| \end{cases} \tag{59}$$

4.3.3. Cálculo do Ponto de Interseção

Sabendo-se que realmente há ponto de interseção e uma vez obtido t_P, podemos calcular o ponto de interseção P fazendo uma interpolação linear (*Lerp*) entre os pontos R e S:

$$P = R + t_P(S - R) \tag{60}$$

Ou de forma vetorial explícita:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} + t_P \left(\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} - \begin{bmatrix} R_x \\ R_y \\ R_z \end{bmatrix} \right)$$

Esta mesma fórmula pode ser escrita com uso dos predicados de Shewchuk. Assim, garantimos que teremos o ponto P que corresponde exatamente ao t_P fornecido, o qual já foi calculado de forma precisa. Convertendo a fórmula acima:

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} R_x + t_P(S_x - R_x) \\ R_y + t_P\big(S_y - R_y\big) \\ R_z + t_P(S_z - R_z) \end{bmatrix}$$

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} t_P(S_x - R_x) - 1(0 - R_x) \\ t_P\big(S_y - R_y\big) - 1\big(0 - R_y\big) \\ t_P(S_z - R_z) - 1(0 - R_z) \end{bmatrix}$$

$$P = \left(\begin{vmatrix} t_P & -R_x \\ 1 & S_x - R_x \end{vmatrix}, \begin{vmatrix} t_P & -R_y \\ 1 & S_y - R_y \end{vmatrix}, \begin{vmatrix} t_P & -R_z \\ 1 & S_z - R_z \end{vmatrix} \right)$$

E, finalmente

$$\begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} orient2d\big((t_P, 0), (1, S_x), (0, R_x)\big) \\ orient2d\left((t_P, 0), \big(1, S_y\big), \big(0, R_y\big)\right) \\ orient2d\big((t_P, 0), (1, S_z), (0, R_z)\big) \end{bmatrix} \tag{61}$$

4.3.4. Ordenação Lexicográfica

Deve-se garantir que a operação de interseção segmento-triângulo obtenha sempre os mesmos pontos de interseção, independentemente da ordem dos vértices do triângulo recebido e da ordem dos pontos extremos do segmento recebido. Por causa de erros numéricos, isto infelizmente não ocorre na prática. Tal comutatividade só pode ser garantida, porém, se todas as operações realizadas no algoritmo de interseção segmento-triângulo forem exatas. No algoritmo proposto, porém, há uma operação de divisão não-exata.

Sendo assim, a inversão dos pontos extremos do segmento possui um efeito bem indesejado. Se, ao invés do segmento

$$e\colon\ Q(t) = R + t(S - R)$$

recebermos o segmento

$$\bar{e}\colon\ Q(s) = S + s(R - S)$$

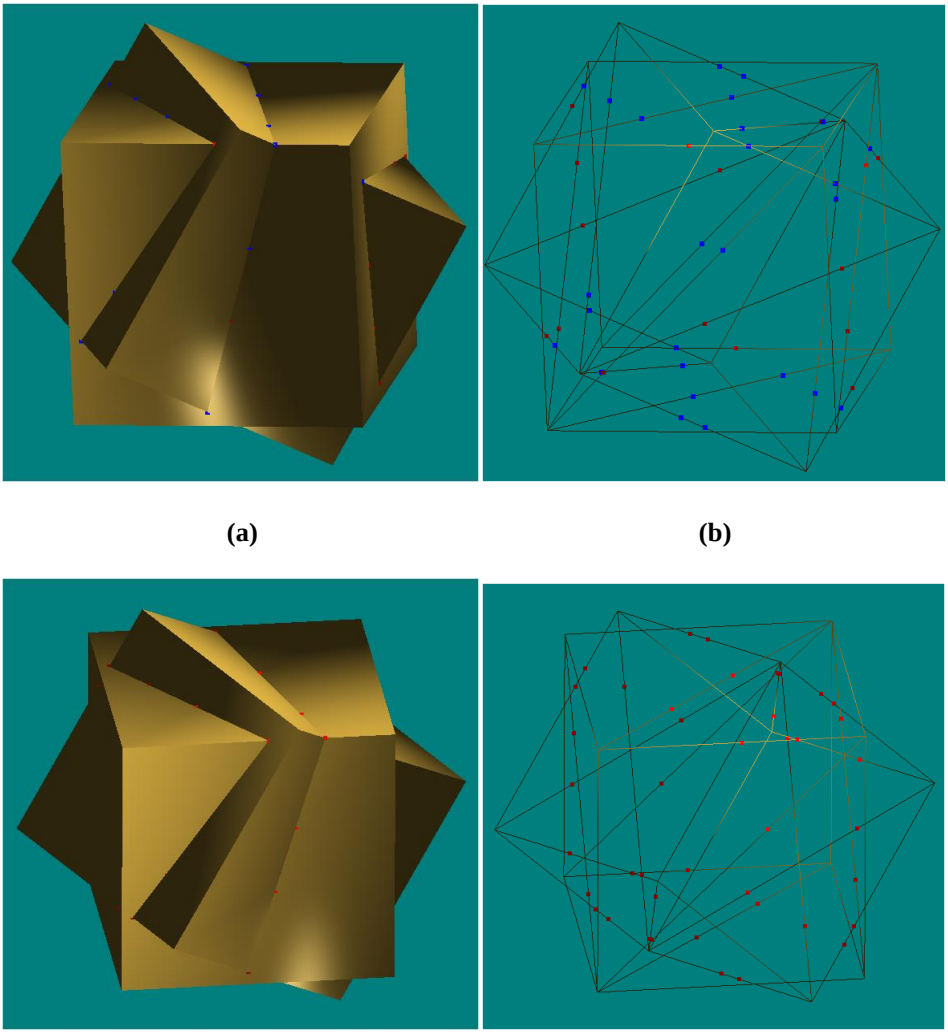
teremos um mesmo ponto de interseção P somente se,

$$s_P = 1 - t_P$$

o que raramente ocorre. Levando-se em conta que t_P e s_P são calculados com uma divisão não-exata, dificilmente eles somarão exatamente 1. Como não há implementações disponíveis da operação de divisão com Aritmética Exata, a solução utilizada será a ordenação lexicográfica.

Assim, ao recebermos o segmento $\bar{e}$, podemos re-invertê-lo para o segmento e, obtendo um ponto idêntico ao que seria dado pela interseção direta com o segmento e.

Para verificar a necessidade da ordenação lexicográfica, foi realizada a interseção entre dois cubos, cada um formado por 12 triângulos, com um deles rotacionado em 45º no eixo X em relação ao outro. Este modelo é mostrado na figura 27.



(a) **(b)**

(c) **(d)**

Figura 27: Testes de Ordenação Lexicográfica

Sem ordenação lexicográfica, há pontos de interseção duplicados e únicos ((a) e (b)), enquanto que, com ordenação lexicográfica, todos os pontos aparecem duplicados ((c) e (d))

De forma bem ingênua, cada triângulo do primeiro cubo teve suas 3 arestas intersectadas com cada triângulo do segundo cubo e vice-versa.

Dessa forma, cada aresta será intersectada duas vezes para cada triângulo. E, mais do que isso, devido à orientação das arestas em cada triângulo, cada aresta será intersectada uma vez em cada sentido.

Obtido todo conjunto de pontos de interseção, cada ponto foi classificado como “duplicado” ou “único”, dependendo da existência de outro ponto com as mesmas coordenadas X, Y e Z. Curiosamente, os pontos únicos aparecem sempre em pares com diferenças da ordem de 1e-15. Na Figura 27, os pontos únicos foram pintados de azul e os duplicados, de vermelho. Primeiro, realizou-se esse teste sem ordenação lexicográfica e, depois com ordenação lexicográfica. Somente com ordenação lexicográfica, todos os pontos apareceram duplicados, como se esperava.

A ordenação lexicográfica irá inverter os pontos extremos de um segmento se:

- A coordenada X do primeiro ponto for maior do que a do segundo ponto extremo;
- Em caso de empate, testa-se se a coordenada Y do primeiro ponto é maior do que a do segundo ponto extremo;
- Em caso de novo empate, testa-se se a coordenada Z do primeiro ponto é maior do que a do segundo ponto extremo.

Caso contrário, mantém-se a ordem original dos pontos extremos do segmento. Em particular, se persistir o empate, sabe-se que o segmento recebido está colapsado. O mesmo critério é utilizado para ordenar os três vértices de um triângulo.

4.3.5. Pseudocódigo

Finalmente, nosso algoritmo de interseção segmento-triângulo pode ser descrito como na Figura 28. Primeiro, fazemos a ordenação lexicográfica dos vértices do triângulo e do segmento recebido. Em seguida, verificamos a orientação dos dois vértices do segmento em relação ao plano do triângulo. Se ambos estes vértices estiverem acima ou abaixo do plano, não haverá interseção. Senão, a menos que ambos os vértices sejam simultaneamente coplanares ao plano do triângulo, verificamos se o possível ponto de interseção se localiza dentro das

bordas do triângulo. Se ele se localizar fora das bordas do triângulo, não haverá interseção. Caso contrário, se o possível ponto de interseção for um vértice, inclua-o na curva de interseção. Se não for um vértice, calcule-o e inclua-o na curva de interseção. Pela estratégia adotada, se ambos os vértices do segmento forem coplanares ao plano do triângulo consideramos que não houve interseção.

```
Segment_Triangle_Intersection(e, T):
Sort all endpoints of e and all vertices of T
If both endpoints of e are over π then
      return false
End If
If both endpoints of e are under π then
      return false
End If
If both endpoints are not coplanar to π then
   If the intersection point lies outside triangle borders
        return false
   End If
   If the intersection point is a vertex of e then
    Add this vertex to curve
    return true
   End If
    Calculate P, the intersection point between e and T
    Add P to curve
    return true
End If
return false
```

Figura 28: Algoritmo Robusto de Interseção Segmento-Triângulo

(Método Segment_Triangle_Intersect)

5

Resultados

5.1. Modelos de Testes Utilizados

A ferramenta de interseção de superfícies desenvolvida neste trabalho foi testada em uma série de modelos, produzindo tanto *chains* como *loops*. A seguir, serão mostrados 6 modelos e os resultados dos seus testes. A seção 5.4 relata testes de eficiência realizados com esses modelos. Para cada modelo, foram realizadas 10 medições de tempo em uma máquina com processador AMD Phenom™ II P920 Quad-Core de 1.6GHz, 4GB de RAM, executando Windows 7, versão 64 bits.

No Modelo A, mostrado na Figura 29, ocorre a interseção entre duas superfícies originalmente representando domos de sal, posicionadas de forma arbitrária no espaço.

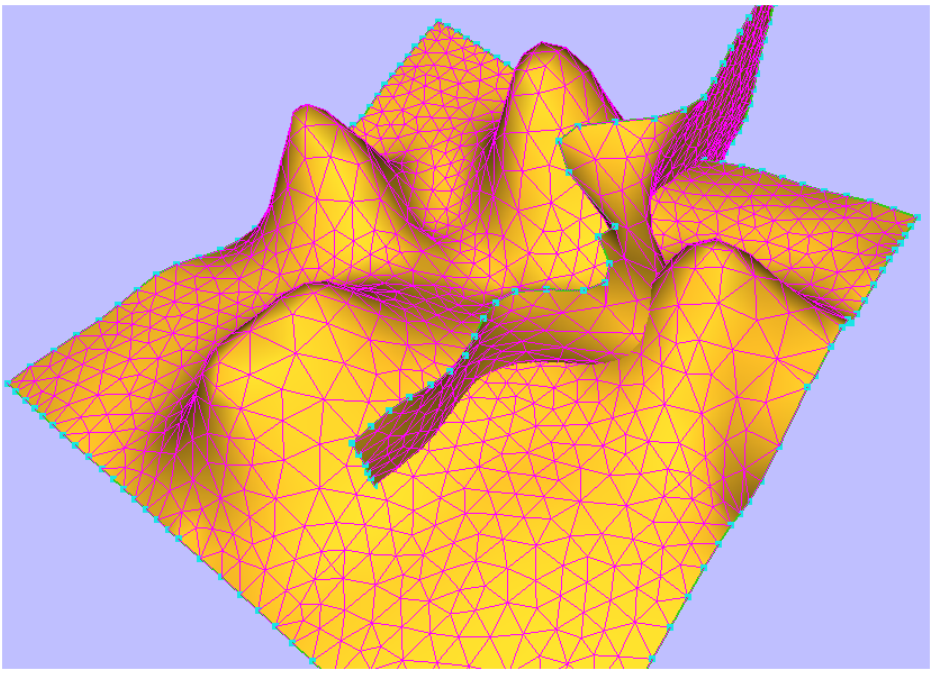


Figura 29: Modelo A: Interseção arbitrária entre superfícies representando domos de sal

Observe, com base na Tabela 2, que, em ambas as superfícies, a área média dos triângulos corresponde a somente 0,0523% da área total da superfície, com o menor triângulo correspondendo a 0,0137% e o maior, a 0,2067%.

Modelo A	Superfície 1	Superfície 2
Área da Superfície	1505377.93	1505377.93
Número de Triângulos	1912	1912
Área média dos Triângulos	787.33	787.33
Mediana da Área dos Triângulos	686.45	686.45
Desvio-Padrão da Área dos Triângulos	380.95	380.95
Área do Menor Triângulo	205.57	205.57
Área do Maior Triângulo	3110.92	3110.92

Tabela 2: Dados do Modelo A

O algoritmo de interseção encontrou, nesse modelo, 252 pares de triângulos em interseção e a curva produzida, retratada na figura 30, teve 127 pontos.

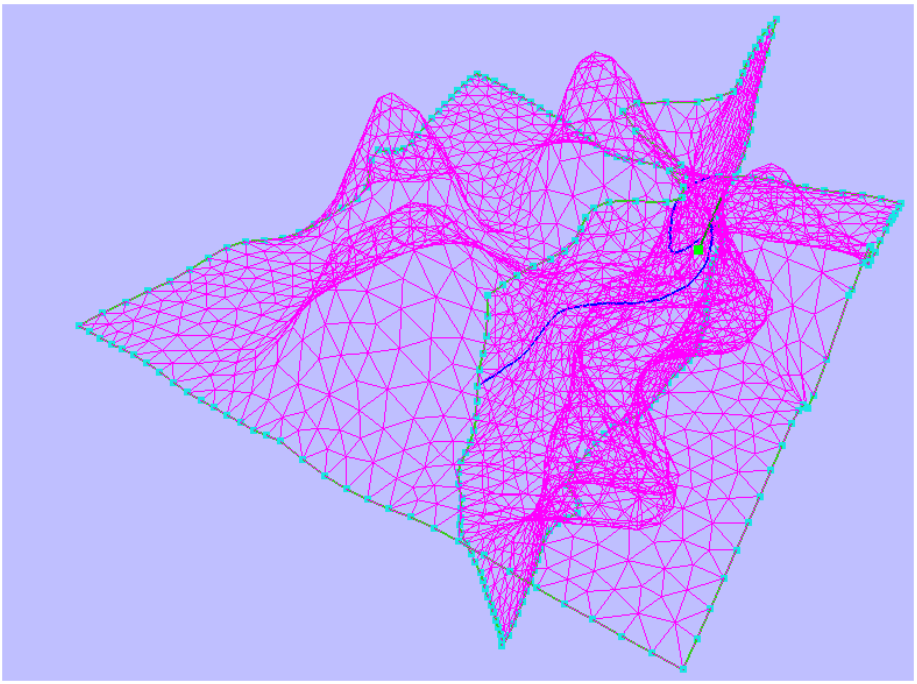


Figura 30: Curva de Interseção Gerada no Modelo A

Já no modelo B, exibido na Figura 31, temos a interseção transversal entre duas falhas.

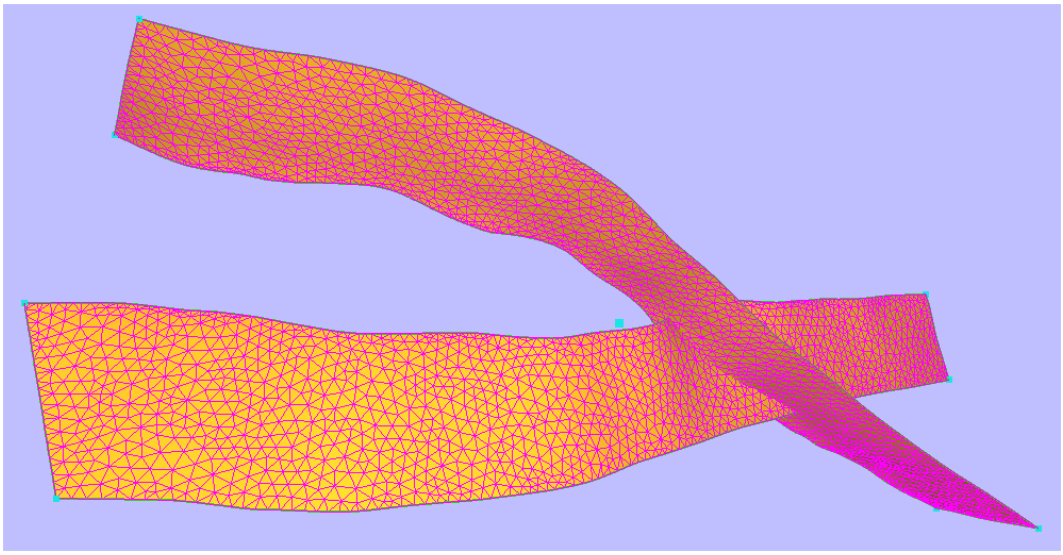


Figura 31: Modelo B: Interseção Transversal entre Falhas

De acordo com a Tabela 3, pode-se observar que, em ambas as superfícies, a área média dos triângulos corresponde a somente 0,0365% da área total da superfície, com o menor triângulo correspondendo a 0,0031% e o maior, a 0,0944%.

Modelo B	Superfície 1	Superfície 2
Área da Superfície	2603160.62	2603160.62
Número de Triângulos	2743	2743
Área média dos Triângulos	949.02	949.02
Mediana da Área dos Triângulos	943.81	943.81
Desvio-Padrão da Área dos Triângulos	340.80	340.80
Área do Menor Triângulo	80.90	80.90
Área do Maior Triângulo	2456.87	2456.87

Tabela 3: Dados do Modelo B

No modelo B, o algoritmo de interseção encontrou 160 pares de triângulos em interseção e a curva produzida teve 81 pontos, como exposto na Figura 32.

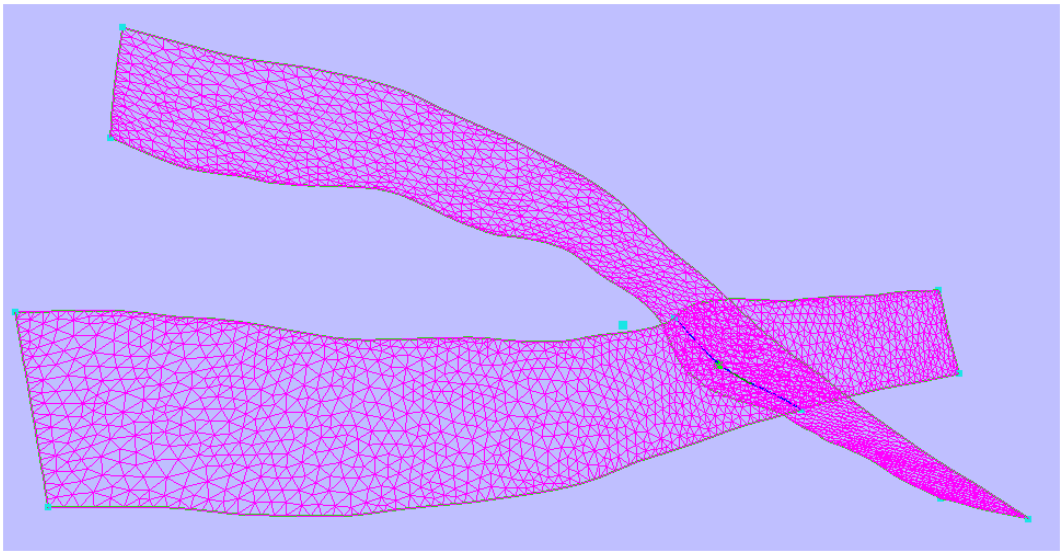


Figura 32: Curva de Interseção Gerada no Modelo B

Por sua vez, o modelo C apresenta duas falhas com interseção longitudinal mostradas na Figura 33.

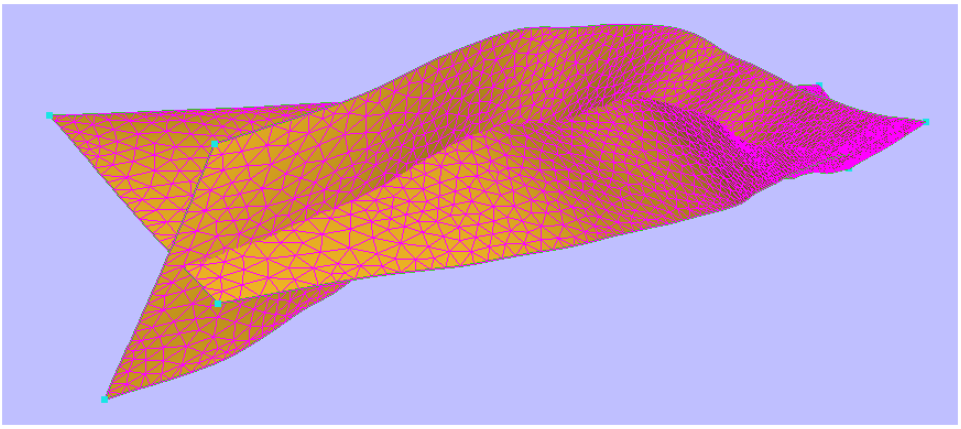


Figura 33: Modelo C: Interseção Longitudinal entre Falhas

Analisando a Tabela 4, pode-se observar que, em ambas as superfícies, a área média dos triângulos corresponde a somente 0,0365% da área total da superfície, com o menor triângulo correspondendo a 0,0031% e o maior, a 0,0944%.

Modelo C	Superfície 1	Superfície 2
Área da Superfície	2603160.62	2603160.62
Número de Triângulos	2743	2743
Área média dos Triângulos	949.02	949.02
Mediana da Área dos Triângulos	943.81	943.81
Desvio-Padrão da Área dos Triângulos	340.80	340.80
Área do Menor Triângulo	80.90	80.90
Área do Maior Triângulo	2456.87	2456.87

Tabela 4: Dados do Modelo C

Nesse modelo, o algoritmo de interseção encontrou 732 pares de triângulos em interseção e a curva produzida, mostrada na figura 34, teve 367 pontos.

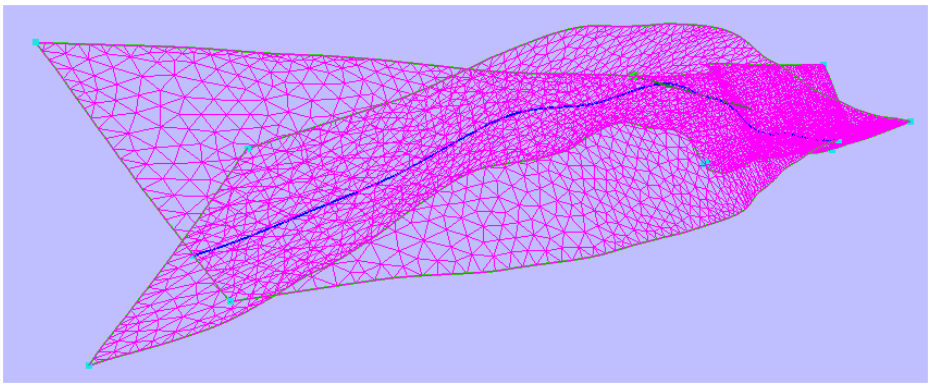
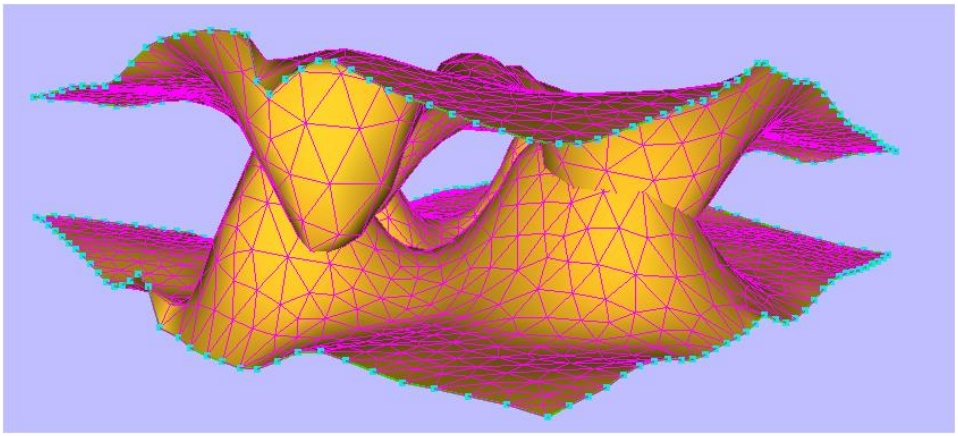
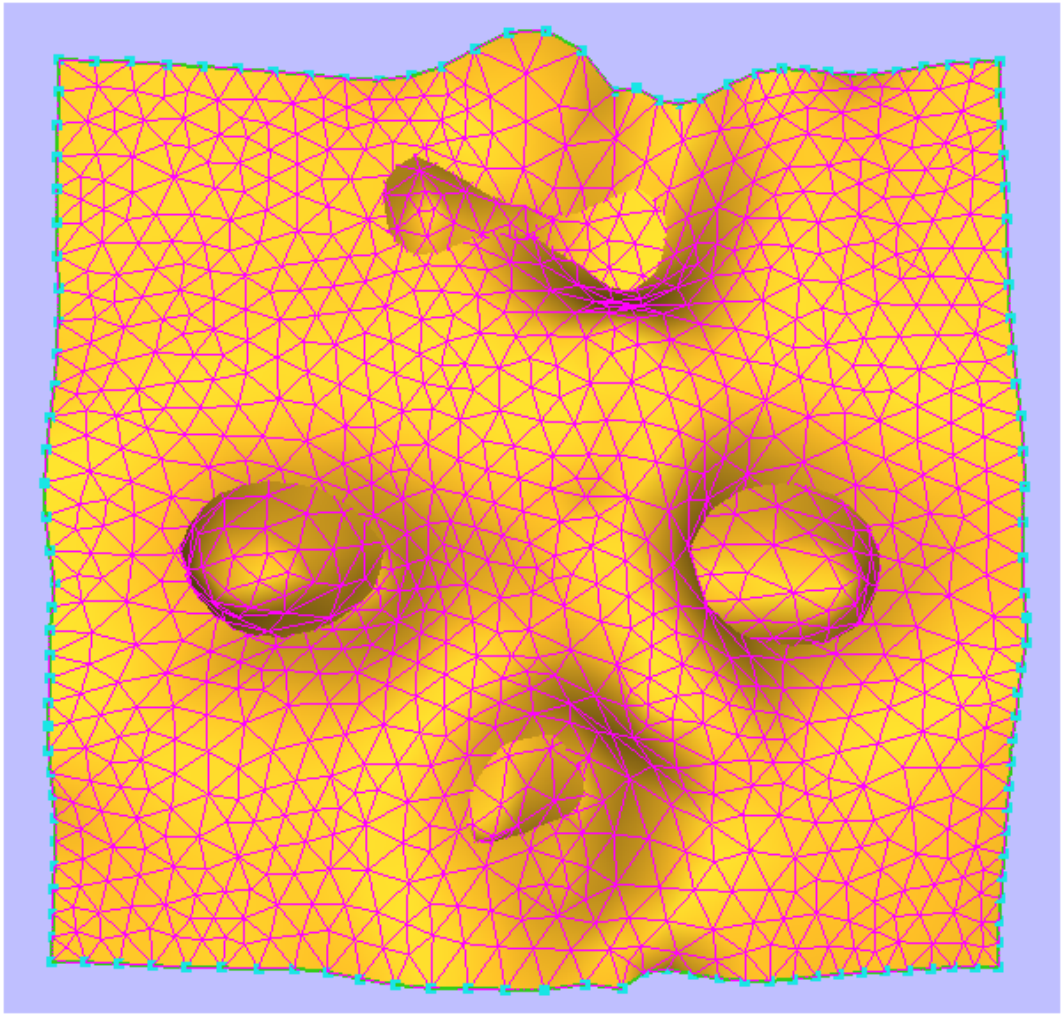


Figura 34: Curva de Interseção Gerada no Modelo C

O Modelo D possui estalagmites e estalactites em interseção, produzindo *loops.* Este modelo é exibido na Figura 35, primeiramente em perspectiva e, depois, em vista ortogonal.



(a)



(b)

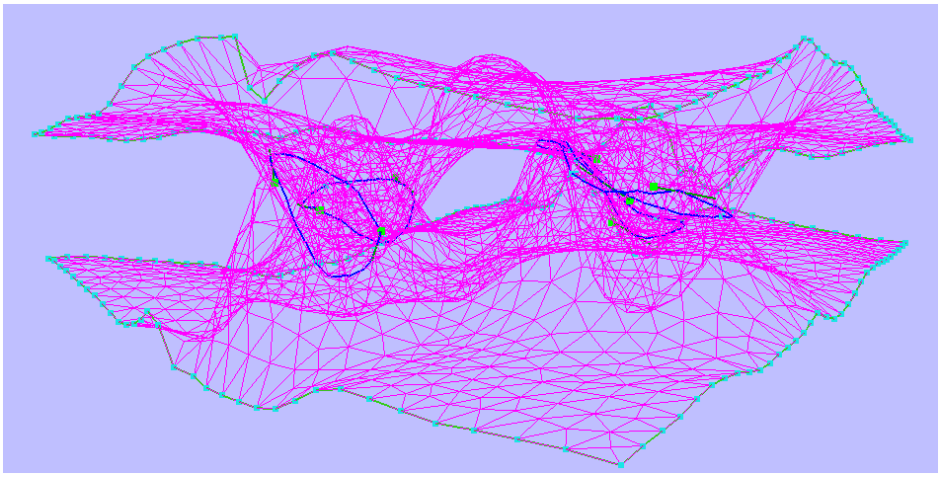
Figura 35: Modelo D: Interseção entre Estalagmites e Estalactites, produzindo loops
(a) Visão em Perspectiva (b) Visão Ortogonal

Agora, observe na tabela 5 que, em ambas as superfícies, a área média dos triângulos corresponde a somente 0,0523% da área total da superfície, com o menor triângulo correspondendo a 0,0137% e o maior, a 0,2067.

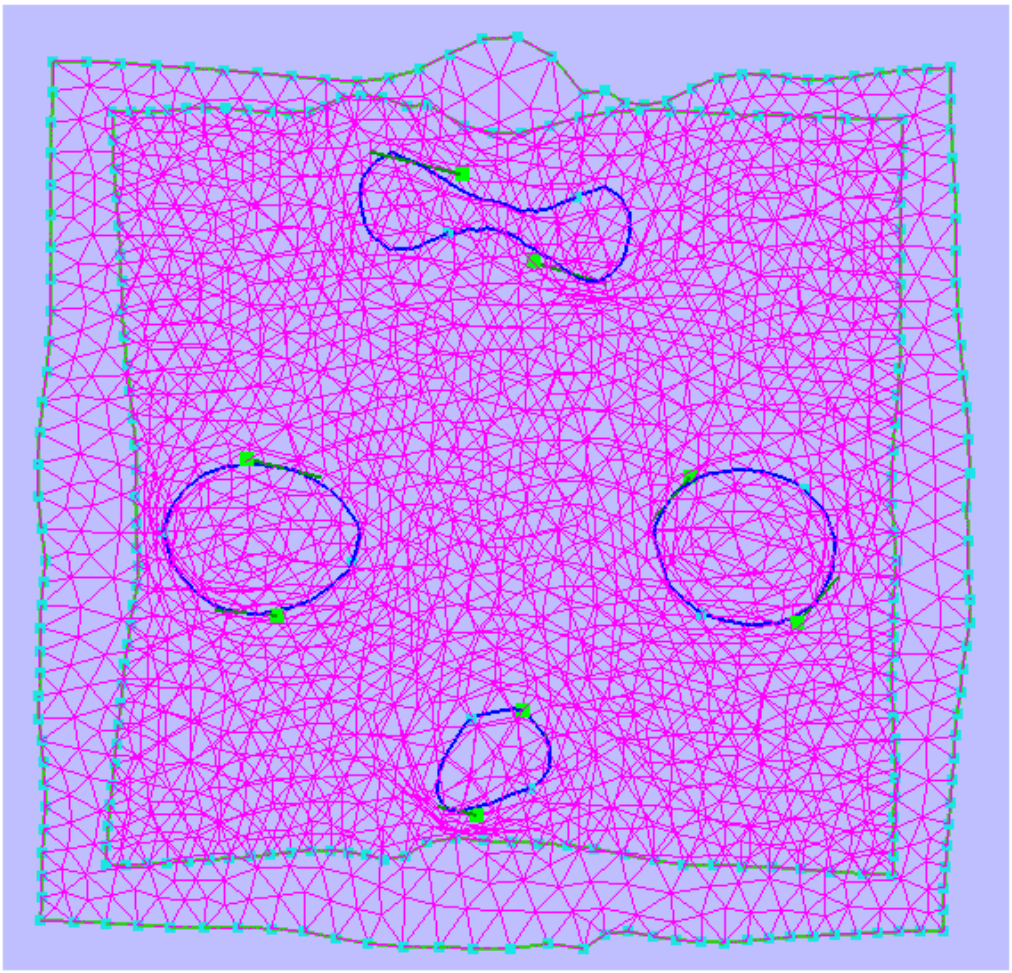
Modelo D	Superfície 1	Superfície 2
Área da Superfície	1505377.93	1505377.93
Número de Triângulos	1912	1912
Área média dos Triângulos	787.33	787.33
Mediana da Área dos Triângulos	686.45	686.45
Desvio-Padrão da Área dos Triângulos	380.95	380.95
Área do Menor Triângulo	205.57	205.57
Área do Maior Triângulo	3110.92	3110.92

Tabela 5: Dados do Modelo D

No modelo D, o algoritmo de interseção encontrou 528 pares de triângulos em interseção e as curvas produzidas tiveram 268 pontos no total. Essas curvas foram retratadas na Figura 36, primeiramente, em perspectiva e , depois, em vista ortogonal.



(a)

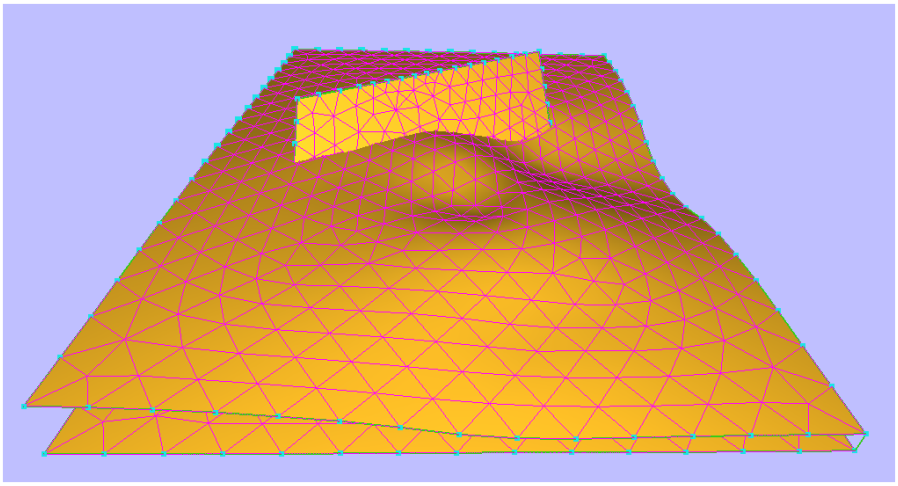


(b)

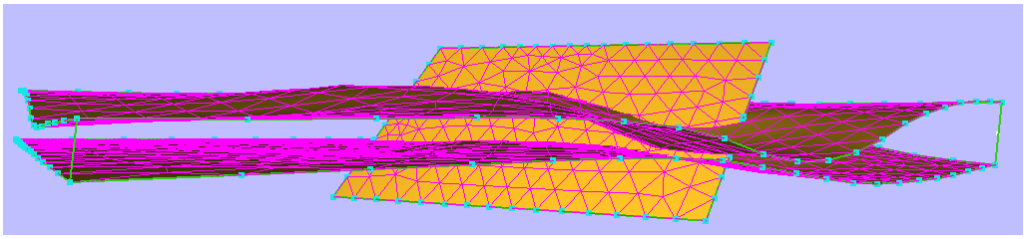
Figura 36: Curvas de Interseção (do tipo *Loop*) Geradas no Modelo D

(a) Visão em Perspectiva (b) Visão Ortogonal

Finalmente, o modelo E, mostrado na Figura 37 em perspectiva e em vista ortogonal, contém uma falha sendo intersectada por dois horizontes.



(a)



(b)

Figura 37: Modelo 20: Interseção de Falha com dois Horizontes

(a) Visão em Perspectiva (b) Visão Ortogonal

Por conveniência, separaremos este modelo em dois sub-modelos: o modelo E.1, compreendendo a falha e o horizonte superior, e o modelo E.2, consistindo na falha e o horizonte inferior. De acordo com a tabela 6, a área média dos triângulos da superfície da falha corresponde a somente 0,3774% da área total dessa superfície, com o menor triângulo correspondendo a 0,1096% e o maior, a 0,6341%. Já para os horizontes, a área média de seus triângulos correspondem a 0,1166% e 0,1124% da área total de suas superfícies, com os menores triângulos correspondendo a 0,0059% e 0,0532% e o maior, a 0,1647% e 0,1545%.

Modelo E	Falha	Horizonte Superior	Horizonte Inferior
Área da Superfície	39903.44	223204.51	221777.58
Número de Triângulos	265	858	890
Área média dos Triângulos	150.58	260.15	249.19
Mediana da Área dos Triângulos	153.61	265.48	260.79
Desvio-Padrão da Área dos Triângulos	37.50	28.68	32.06
Área do Menor Triângulo	43.75	131.23	118.19
Área do Maior Triângulo	253.02	367.68	342.54

Tabela 6: Dados do Modelo E

No modelo E, o algoritmo de interseção encontrou, respectivamente, 108 e 114 pares de triângulos em interseção e as curvas produzidas tiveram, respectivamente, 55 e 58 pontos. Essas curvas estão sendo exibidas na Figura 38.

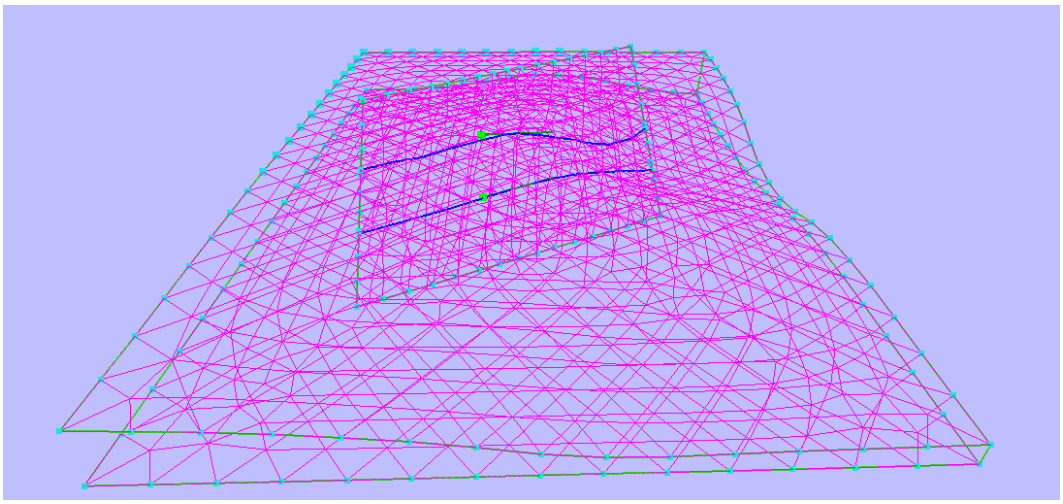
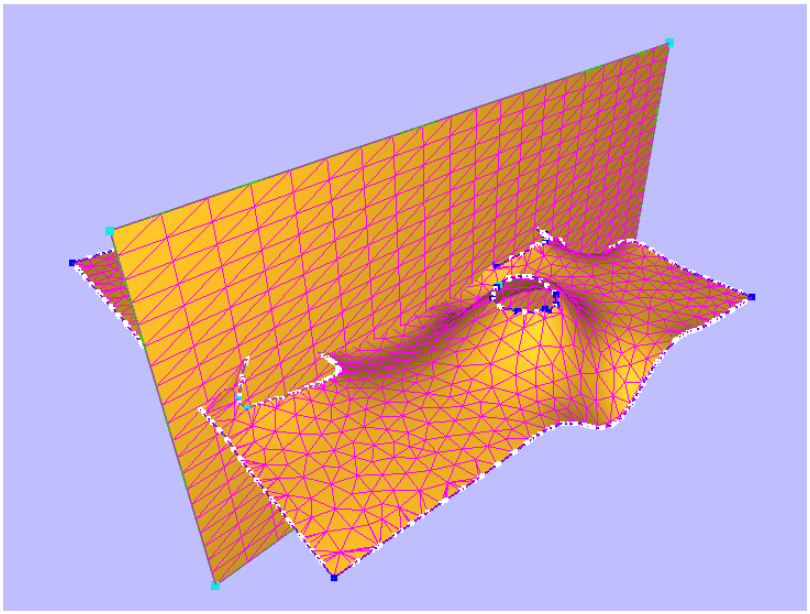


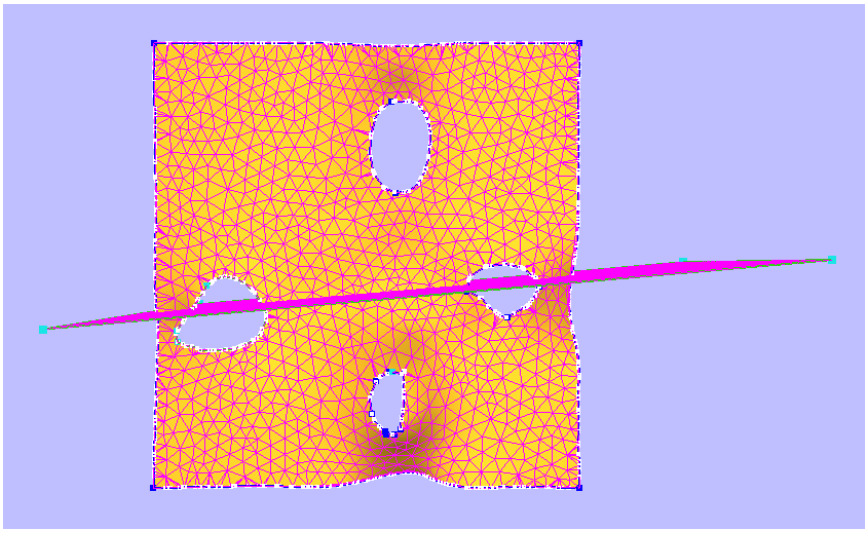
Figura 38: Curvas de Interseção (do tipo *Chain*) Geradas no Modelo E

5.2. Testes de Confiabilidade

O algoritmo proposto se mostrou confiável em todos os testes realizados. Em particular, uma entrada potencialmente hostil a um algoritmo de interseção de superfícies triangulares são superfícies com triângulos faltando. Isto pode ocorrer tanto de forma acidental, com pequenos furos em forma de triângulos dispersos por toda a superfície, como de forma proposital, no caso de grandes áreas da superfície sem nenhum triângulo, representando *genera*(buracos) topológicos.



(a)



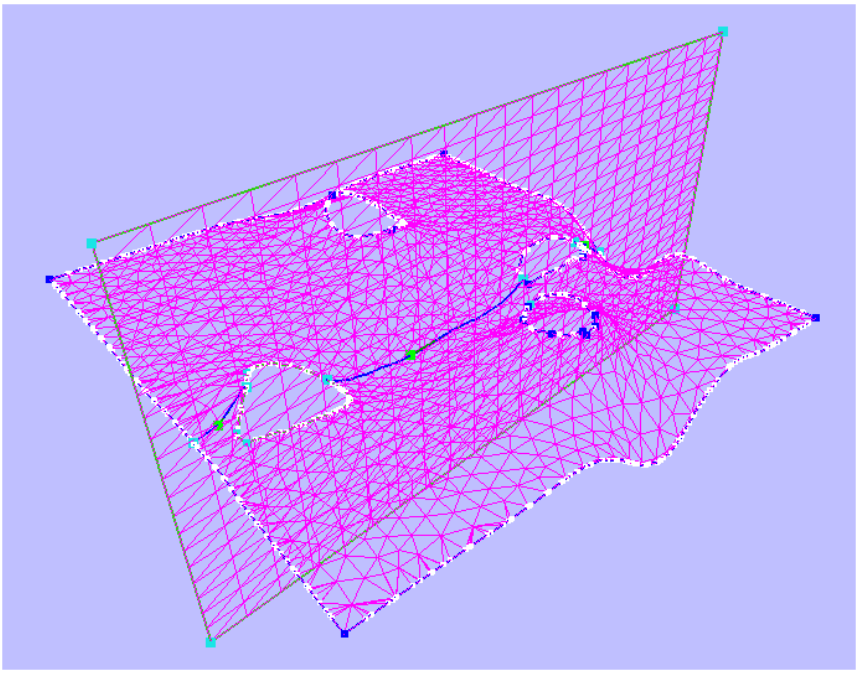
(b)

Figura 39: Interseção de uma superficie triangular plana com a superficie triangular de um relevo contendo 4 buracos: (a) em perspectiva e (b) em projeção ortogonal

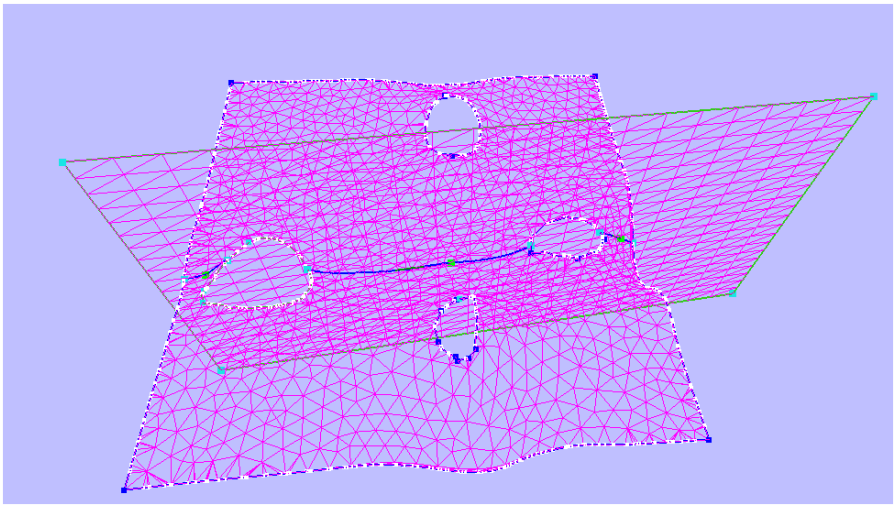
Para testar a confiabilidade do algoritmo proposto a estes casos, foi realizada a interseção de uma superfície triangular plana com uma superfície triangular de um relevo, dotada de quatro buracos, como mostrado na Figura 39. A superfície plana está direcionada de modo que cruze dois buracos de uma única vez.

Ao se realizar a interseção entre estas superfícies, a curva de interseção é interrompida, ao encontrar um buraco, e, assim que sai dele, é retomada. Isto ocorre com os dois buracos, como pode ser visto na Figura 40.

Seria um erro aceitar uma curva de interseção que possuisse um trecho ligando dois pontos na borda do buraco. Observe que simplesmente considerando as “bordas do buraco” como “bordas da superfície”, o algoritmo produz curvas de interseção da maneira desejada. Ou seja, tratar buracos torna-se trivial, uma vez que já se trata múltiplas curvas de interseção (especificamente do tipo *chain*).



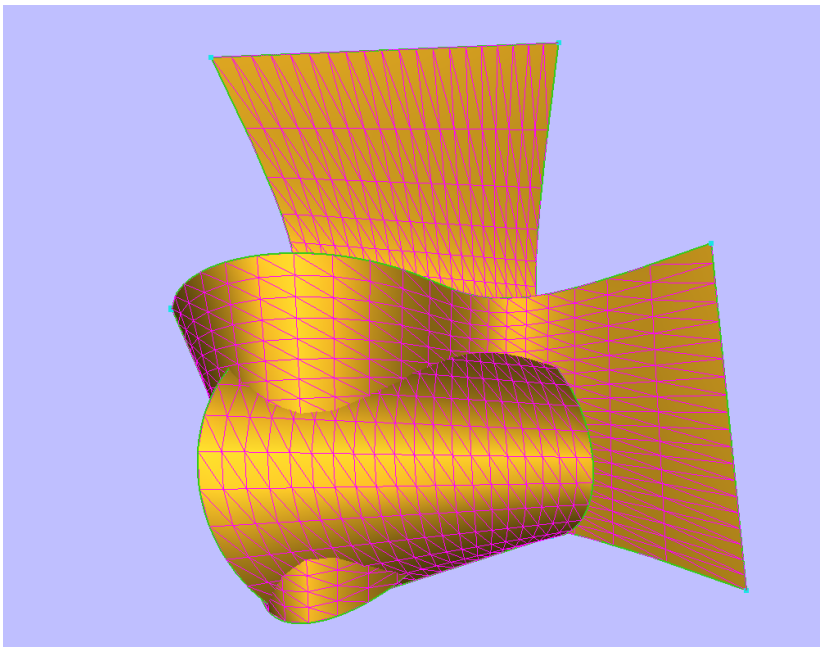
(a)



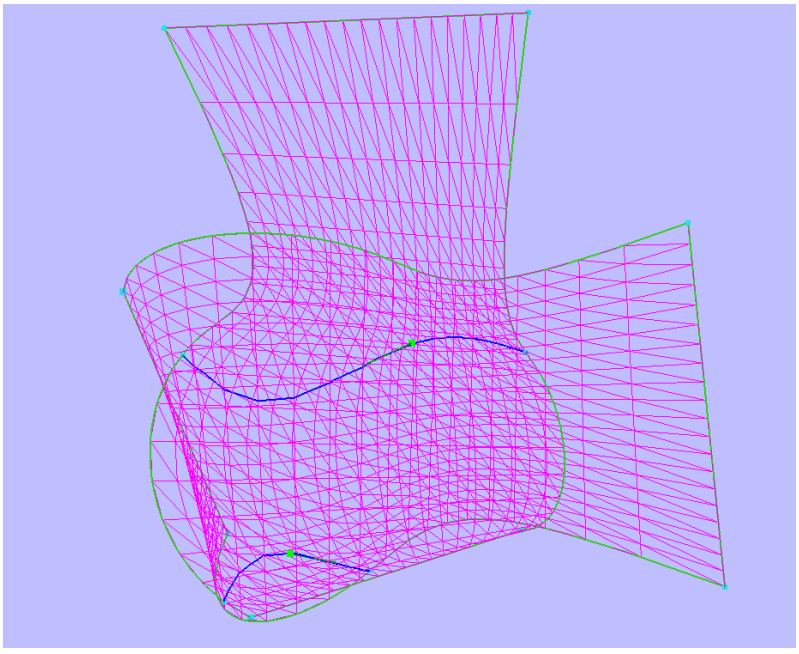
(b)

Figura 40: Curvas de Interseçao Geradas no Modelo com Buracos: (a) em perspectiva e (b) em projeção ortogonal.

A Figura 41 mostra que o algoritmo proposto também funciona para o caso de duas superfícies produzindo mais de uma curva de interseção do tipo *chain*.



(a)

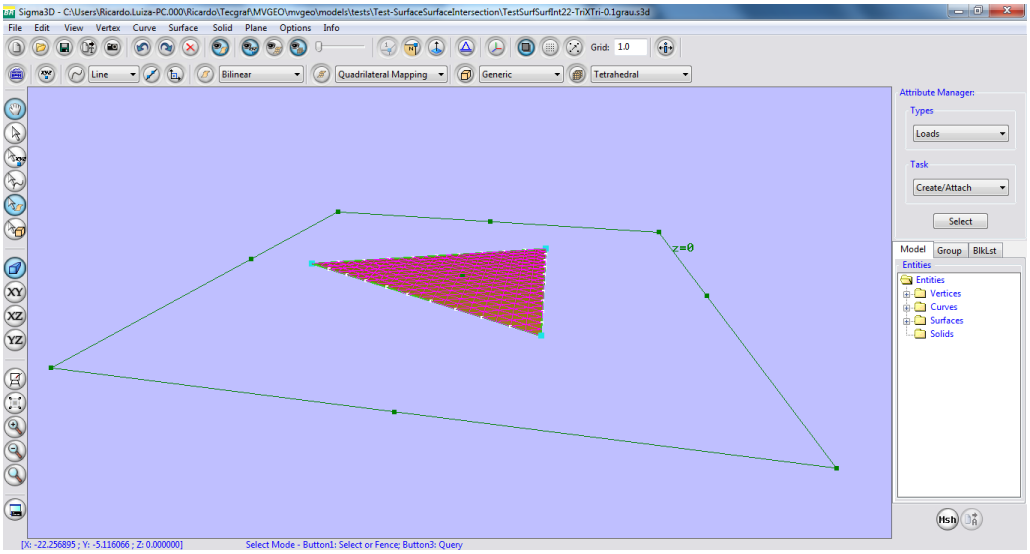


(b)

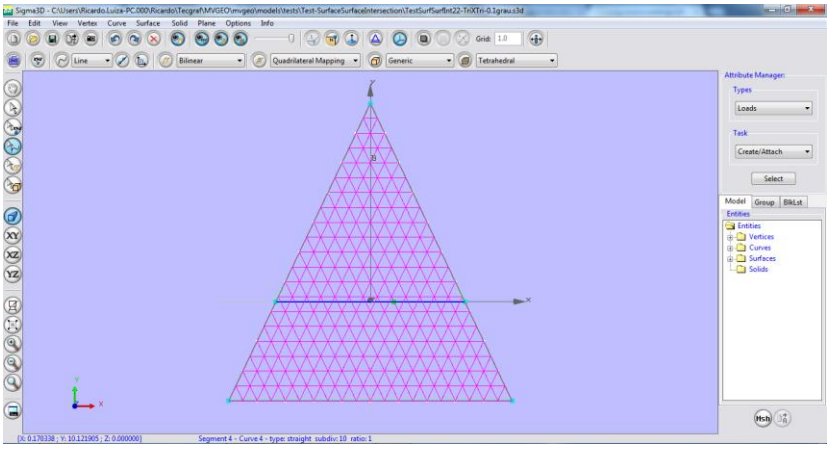
Figura 41: Duas superfícies triangulares em formatos senoidais (a) e suas duas curvas de interseção de tipo *chain* (b)

Também foi confeccionado um teste que explicita o problema de robustez. Foram criadas duas superfícies triangulares, em formato de

triângulo, quase coplanares. A saber, elas estão rotacionadas em 0.1 grau em relação ao eixo X. Este modelo é mostrado na Figura 42.



(a)



(b)

Figura 42: (a) Modelo de Teste das Superfícies Quase Coplanares; (b) A curva de interseção gerada com Aritmética Exata

Realizando a interseção com *back-face culling* e com Aritmética Exata foi criada uma curva de interseção com 27 pontos, sem problemas. Já realizando a mesma operação sem Aritmética Exata, ocorre um erro fatal no programa. Investigando mais a fundo, em modo *debug*, verificou-se que, sem aritmética exata, foram geradas 5 curvas com 32 pontos no total, apresentando interrupções e zigue-zagues.

Comparado o valor dos predicados calculados com e sem Aritmética Exata, verificou-se que, sem Aritmética Exata, alguns bons

pontos de interseção foram recusados, por possuírem coordenadas baricêntricas de valores negativos bem baixos. Por exemplo, o primeiro ponto de interseção, de coordenadas

$$\begin{pmatrix} 6.66666666666666 \\ 2.86807614694833E-17 \\ 0.00000000000000 \end{pmatrix}$$

possui, quando calculado com Aritmética Exata, as seguintes coordenadas baricêntricas:

$$\begin{pmatrix} u_{exact} \\ v_{exact} \\ w_{exact} \end{pmatrix} = \frac{1}{0.00192422452340288} \begin{pmatrix} 0.00128281634893525 \\ 0.00000000000000 \\ 0.00128281634893525 \end{pmatrix}$$

Enquanto que, sem Aritmética Exata, suas coordenadas baricêntricas são:

$$\begin{pmatrix} u_{inexact} \\ v_{inexact} \\ w_{inexact} \end{pmatrix} = \frac{1}{0.00192422452340288} \begin{pmatrix} 0.00128281634893525 \\ -1.100000000000E-19 \\ 0.00128281634893525 \end{pmatrix}$$

Como, sem Aritmética Exata, a segunda coordenada baricêntrica resulta em um número negativo, este ponto de interseção é descartado. O mesmo não ocorre na presença de Aritmética Exata, visto que esta coordenada será precisamente zero. Este teste demonstra como os maus descartes são capazes de adulterar o correto prosseguimento da busca por interseções, produzindo curvas de interseção caóticas.

Na verificação da interseção segmento-plano do triângulo, ocorreram minúsculas diferenças entre os predicados com e sem Aritmética Exata, o que não foi suficiente para adulterar os resultados dos testes de descarte. Nesta etapa do algoritmo, também se comparou os predicados de Aritmética Exata Adaptativo e Não-Adaptativo. Nenhuma diferença foi encontrada, o que demonstra que os predicados adaptativos são tão confiáveis quanto os predicados não-adaptativos.

5.3. Testes de Precisão

Os testes preliminares realizados na interseção de dois cubos (com dois triângulos em cada face do cubo) rotacionados sem Aritmética Exata em 1º, 2º e 45º, mostrou que a unicidade de pontos de interseção (dois pontos de interseção iguais em teoria e na prática) só pode ser obtida com ordenação lexicográfica, mesmo quando se utilizou aritmética exata

no cálculo de interseção. Uma explicação para esse fato é que a divisão, sem ordenação topológica, poderia introduzir erro numérico. Com essa ordenação, o erro numérico cometido é o mesmo para todos os pontos de interseção teoricamente iguais. Por outro lado, a aritmética exata altera as últimas casas decimais dos pontos de interseção, garantindo uma precisão um pouco maior.

O algoritmo de interseção de superfícies proposto foi testado, quanto a sua precisão, em duas versões: uma utilizando predicados de aritmética exata e outra utilizando os mesmos predicados implementados com aritmética tradicional. Duas medições de precisão foram realizadas:

- de Coplanaridade, avaliando o módulo do volume do tetraedro formado pelos 3 pontos do triângulo, além do ponto de interseção obtido.
- de Colinearidade, avaliando o módulo do produto vetorial(área) entre o vetor diretor do segmento e um vetor do primeiro extremo do segmento para o ponto de interseção obtido.

Essas medições foram feitas com Aritmética Exata, independentemente da forma como se calculou os pontos de interseção (com ou sem Aritmética Exata). Idealmente, ambos esses valores devem ser zero. Portanto, quanto mais próximo de zero, mais preciso foi o cálculo.

Os erros de coplanaridade e colinearidade de cada ponto de interseção, em cada modelo, são mostrados nos gráficos numerados como Figuras de 43 a 54. Nesses gráficos, as curvas nomeadas com o termo “*exact*” referem-se a pontos calculados com Aritmética Exata e, as com o termo “*inexact*”, aos calculados sem aritmética exata. Cada gráfico mostra os erros de todos os pontos de interseção de um modelo. Nas Tabelas de número 7 a 12, extraíram-se, desses erros numéricos, sua soma (erro acumulado), média, mediana, desvio-padrão, valor mínimo e valor máximo.

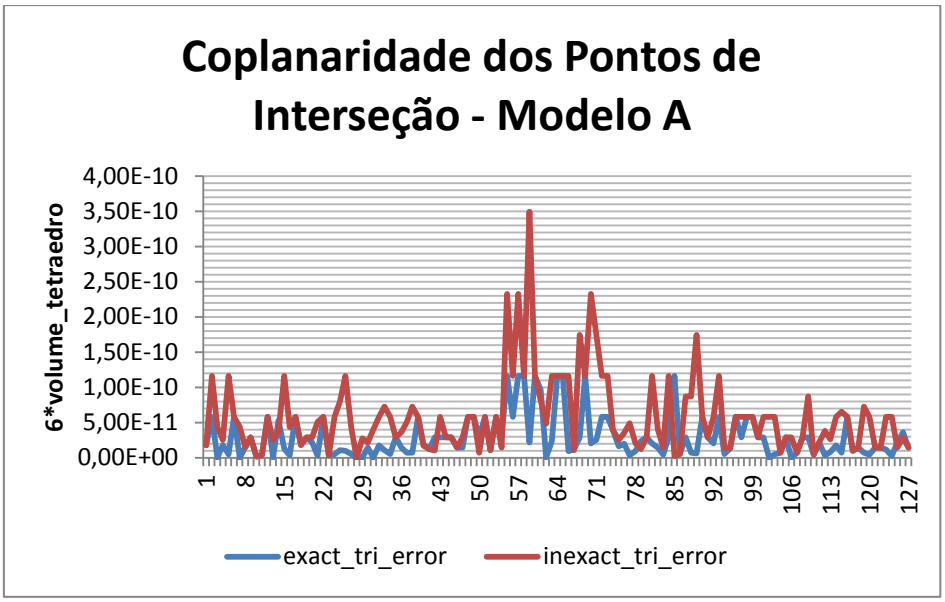


Figura 43: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo A

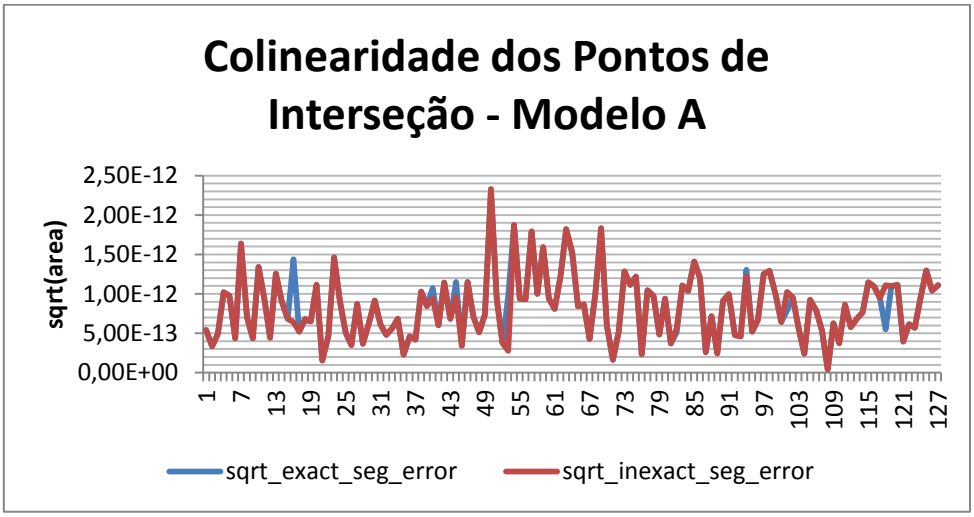


Figura 44: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo A

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	3.63E-09	7.40E-09	1.10E-22	1.08E-22
MEDIA	2.86E-11	5.82E-11	8.64E-25	8.49E-25
MEDIANA	1.82E-11	4.87E-11	7.07E-25	7.07E-25
DESVPAD	2.99329E-11	5.43451E-11	8.38391E-25	8.32306E-25
MIN	7.12E-15	7.12E-15	1.88E-27	1.88E-27
MAX	1.16E-10	3.49E-10	5.43E-24	5.43E-24

Tabela 7: Resultados dos Testes de Precisão do Modelo A

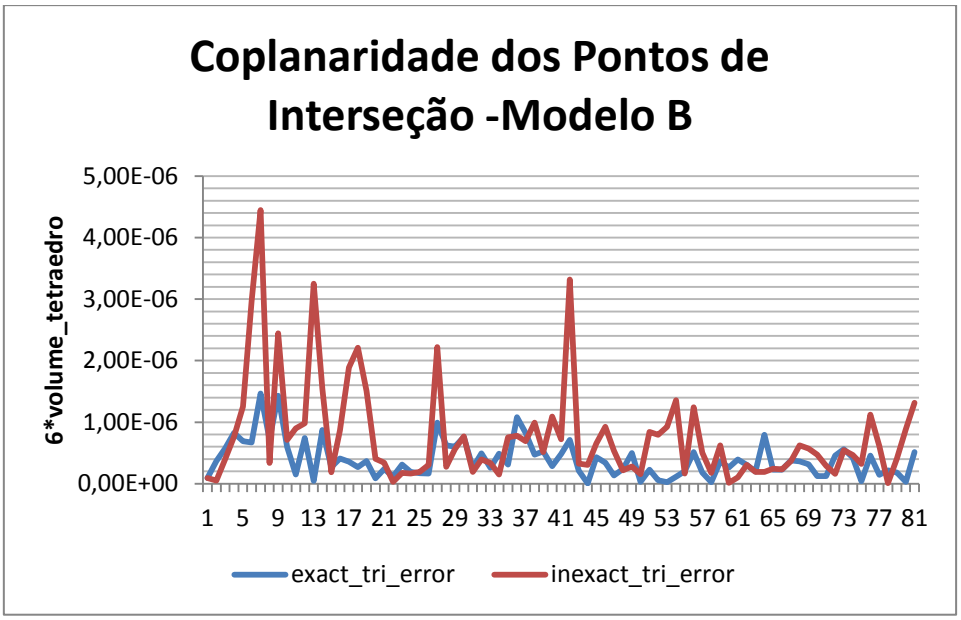


Figura 45: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo B

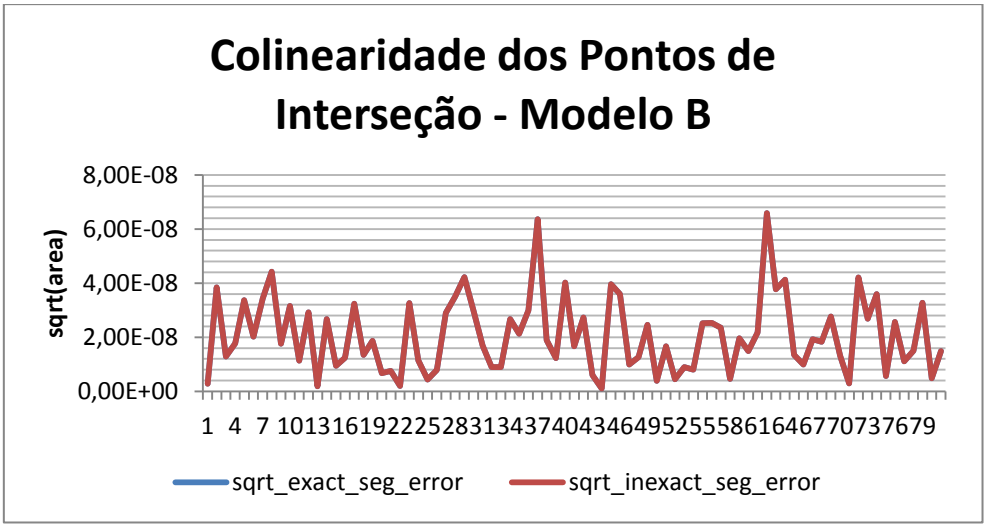


Figura 46: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo B

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	3.17E-05	6.16E-05	5.00E-14	5.00E-14
MEDIA	3.17E-07	5.12E-07	3.37E-16	3.37E-16
MEDIANA	3.17E-07	5.12E-07	3.37E-16	3.37E-16
DESVPAD	2.98144E-07	8.19426E-07	7.77401E-16	7.77401E-16
MIN	4.74E-09	1.02E-08	1.07E-18	1.07E-18
MAX	1.47E-06	4.45E-06	4.33E-15	4.33E-15

Tabela 8: Resultados dos Testes de Precisão do Modelo B

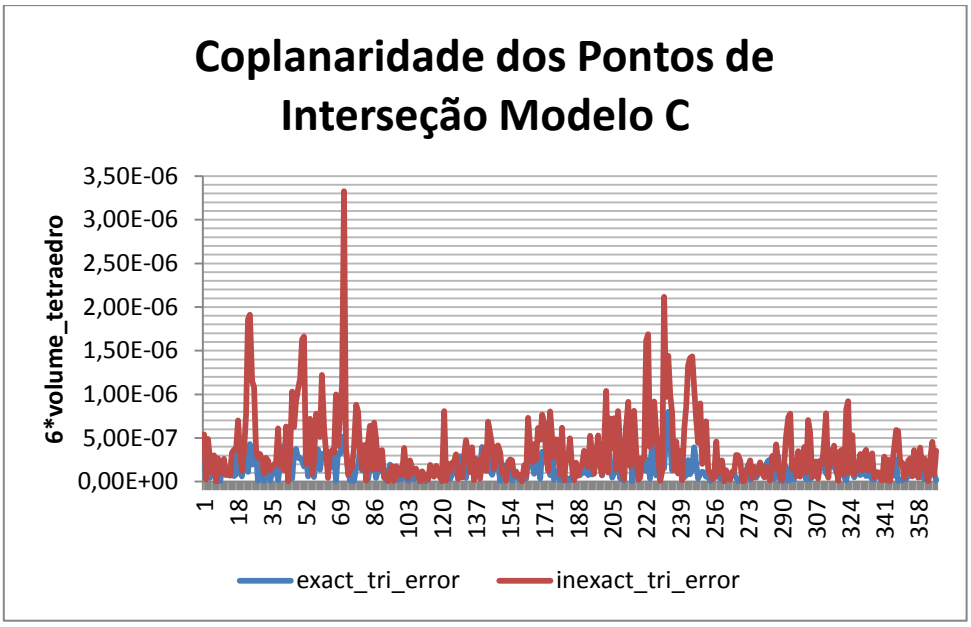


Figura 47: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo C

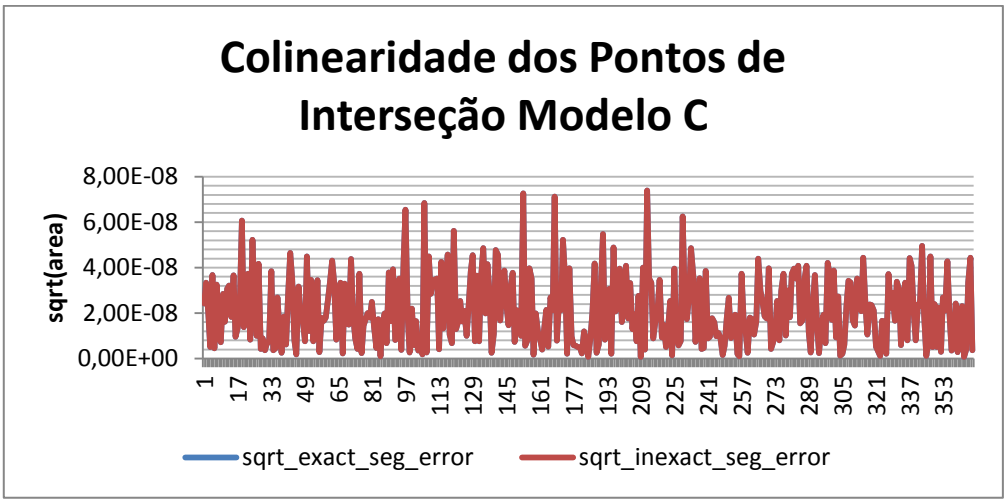


Figura 48: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo C

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	5.60E-05	1.28E-04	2.46E-13	2.46E-13
MEDIA	1.52E-07	3.50E-07	6.71E-16	6.71E-16
MEDIANA	1.17E-07	2.39E-07	3.30E-16	3.30E-16
DESVPAD	1.31293E-07	3.85009E-07	8.60815E-16	8.60815E-16
MIN	2.91E-11	3.20E-10	4.48E-19	4.48E-19
MAX	7.99E-07	3.33E-06	5.46E-15	5.46E-15

Tabela 9: Resultados dos Testes de Precisão do Modelo C

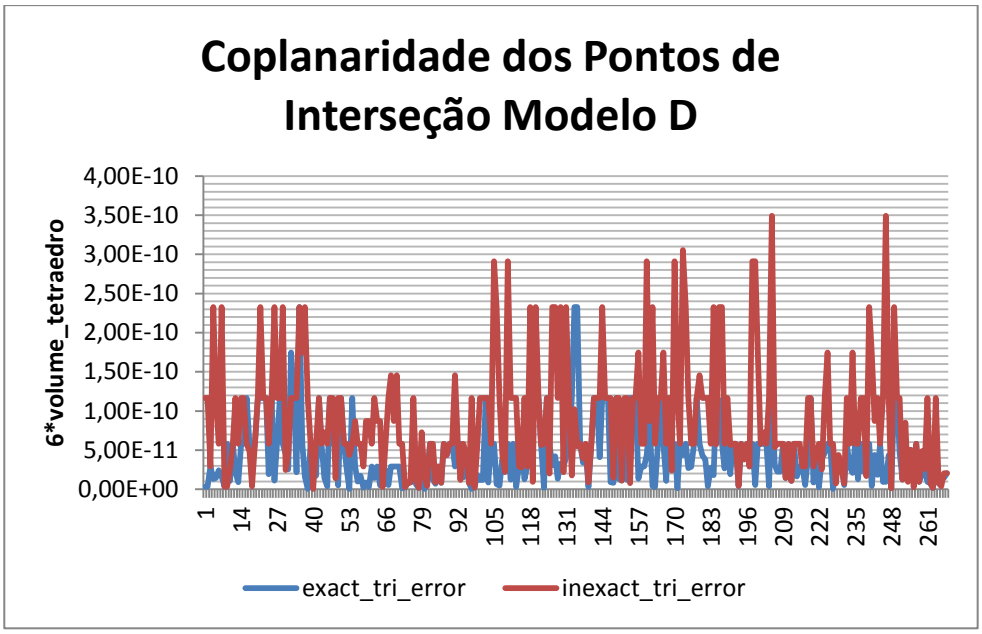


Figura 49: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo D

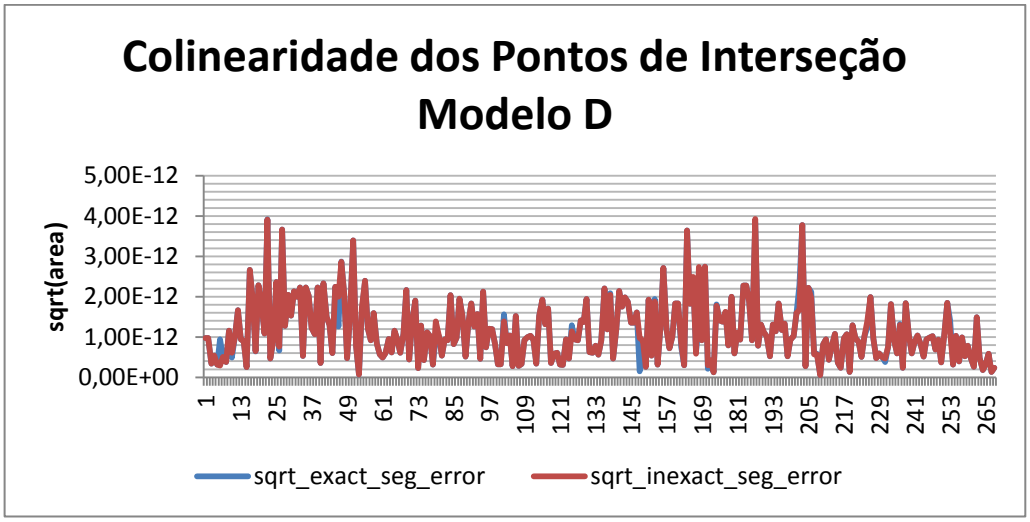


Figura 50: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo D

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	1.20E-08	2.47E-08	4.96E-22	5.02E-22
MEDIA	4.48E-11	9.21E-11	1.85E-24	1.87E-24
MEDIANA	2.91E-11	5.82E-11	9.65E-25	9.65E-25
DESVPAD	4.16432E-11	7.31801E-11	2.45165E-24	2.47931E-24
MIN	3.07E-13	3.78E-13	2.74E-27	2.74E-27
MAX	2.33E-10	3.49E-10	1.54E-23	1.54E-23

Tabela 10: Resultados dos Testes de Precisão do Modelo D

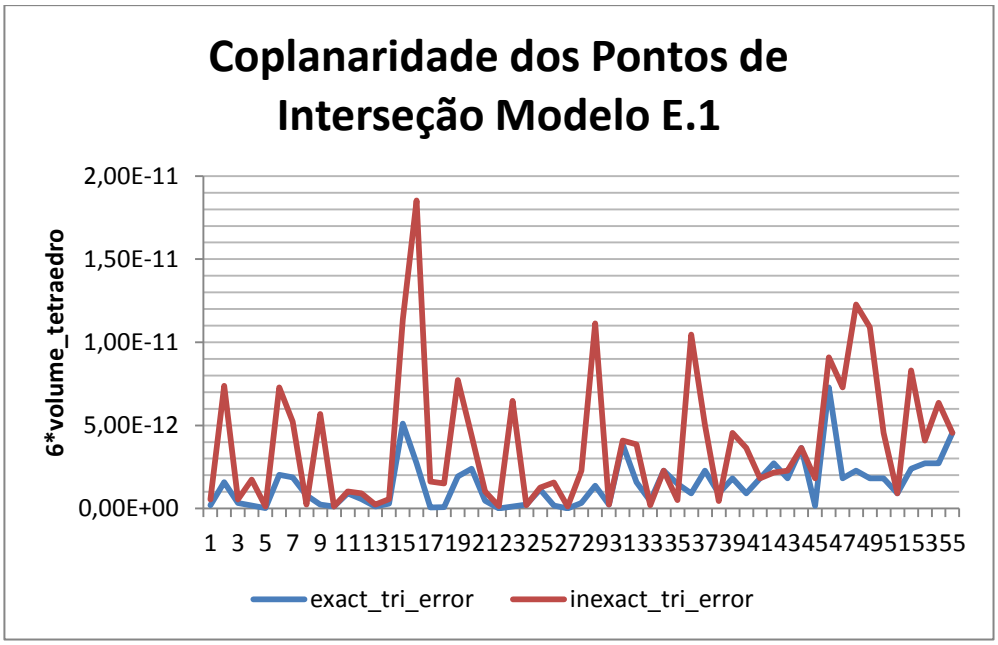


Figura 51: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo E.1

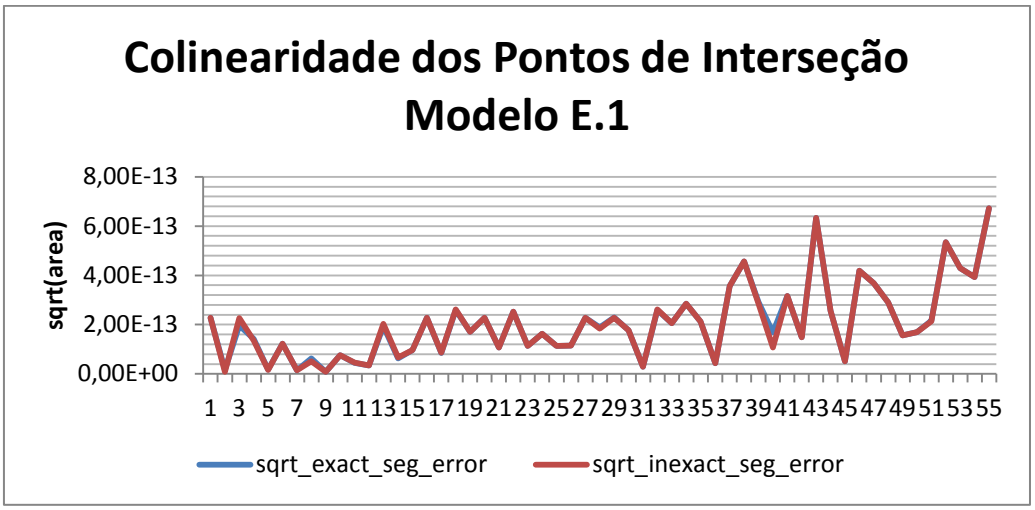


Figura 52: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo E.1

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	8.05E-11	2.16E-10	3.56E-24	3.55E-24
MEDIA	1.46E-12	3.93E-12	6.47E-26	6.46E-26
MEDIANA	1.14E-12	2.27E-12	3.56E-26	3.39E-26
DESVPAD	1.46E-12	4.00E-12	9.25E-26	9.26E-26
MIN	7.11E-15	1.14E-13	6.84E-29	6.84E-29
MAX	7.28E-12	1.85E-11	4.52E-25	4.52E-25

Tabela 11: Resultados dos Testes de Precisão do Modelo E.1

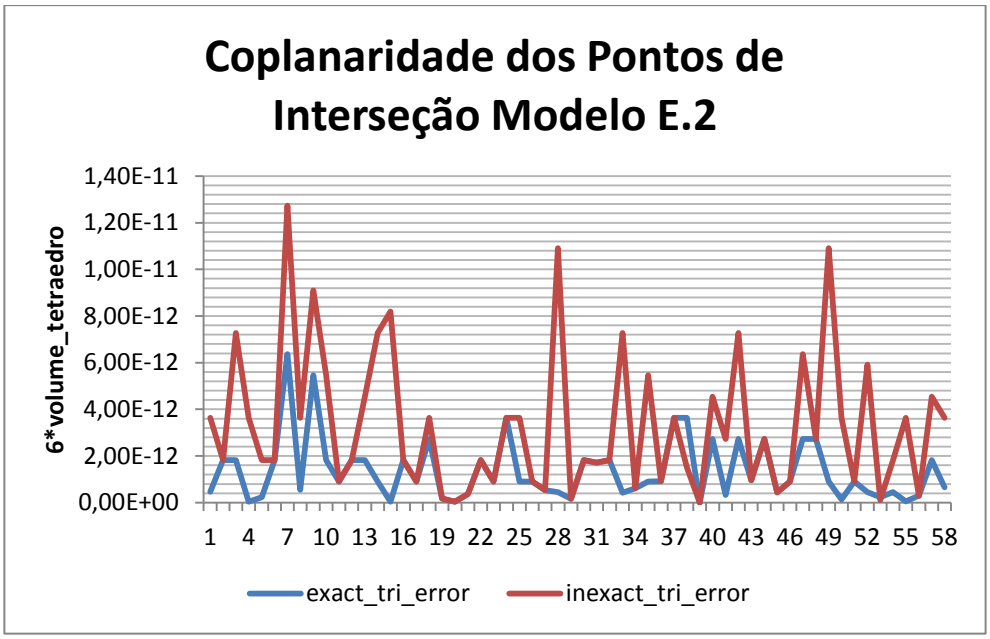


Figura 53: Gráfico do teste de Coplanaridade dos Pontos de Interseção do Modelo E.2

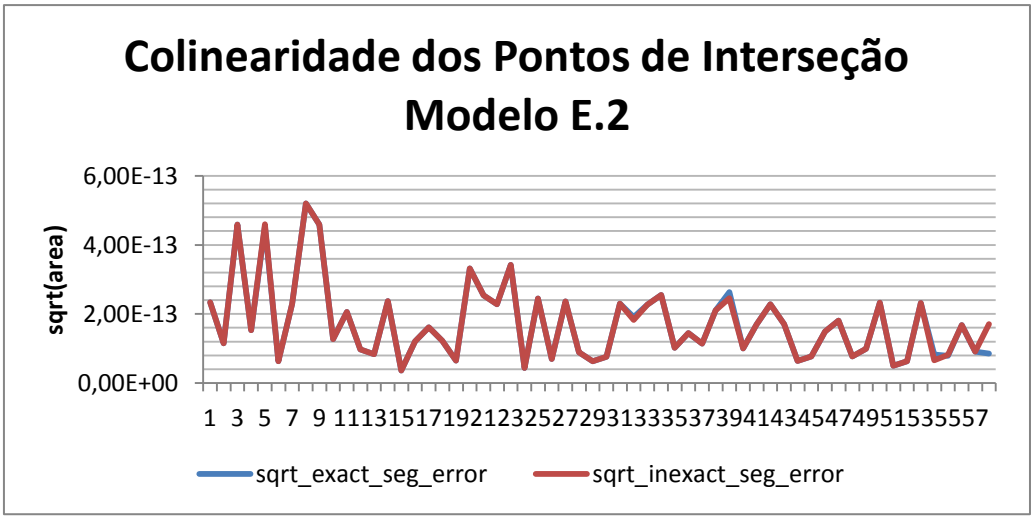


Figura 54: Gráfico do teste de Colinearidade dos Pontos de Interseção do Modelo E.2

Medida	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata
SOMA	7.90E-11	1.92E-10	2.46E-24	2.47E-24
MEDIA	1.36E-12	3.31E-12	4.23E-26	4.26E-26
MEDIANA	9.09E-13	2.27E-12	2.29E-26	2.48E-26
DESVPAD	1.32E-12	3.02E-12	5.68E-26	5.70E-26
MIN	1.44E-15	1.44E-15	1.29E-27	1.29E-27
MAX	6.37E-12	1.27E-11	2.71E-25	2.71E-25

Tabela 12: Resultados dos Testes de Precisão do Modelo E.2

Nos gráficos mostrados anteriormente, pode-se observar que há raros momentos em que a Aritmética Exata fornece valores ligeiramente menos precisos do que quando não é utilizada. Na realidade, sem aritmética exata, ocorre truncamento de valores, o que pode, acidentalmente, tornar um valor "mais preciso". Assim, o valor arredondado é, por acaso, mais preciso do que seu valor verdadeiro. Por causa disto, em alguns raros testes de colinearidade, os resultados sem Aritmética Exata parecem mais precisos do que os com Aritmética Exata.

Convém ressaltar também, que, apesar do uso da Aritmética Exata, não foi possível zerar todos os erros numéricos sempre. Isto se deve ao fato que a divisão continuou inexata e tem um peso grande no cálculo dos pontos de interseção. Ainda assim, pode-se observar que, na versão com Aritmética Exata, houve mais ocorrências de erros de coplanaridade zerados do que na versão sem Aritmética Exata.

A Tabela 13 consolida os erros acumulados para todos os pontos de uma mesma curva de interseção em diferentes modelos.

Modelo	6*Volume com Aritmética Exata	6*Volume sem Aritmética Exata	4*Quadrado da Área com Aritmética Exata	4*Quadrado da Área sem Aritmética Exata	Diferença dos Volumes	Diferença das Áreas
Modelo A	3.63E-09	7.40E-09	1.10E-22	1.08E-22	51%	-1.73%
Modelo B	3.17E-05	6.16E-05	5.00E-14	5.00E-14	49%	0.00%
Modelo C	5.60E-05	1.28E-04	2.46E-13	2.46E-13	56%	0.00%
Modelo D	1.20E-08	2.47E-08	4.96E-22	5.02E-22	51%	1.17%
Modelo E.1	8.05E-11	2.16E-10	3.56E-24	3.55E-24	63%	-0.20%
Modelo E.2	7.90E-11	1.92E-10	2.46E-24	2.47E-24	59%	0.58%
				Média	55%	-0.03%

Tabela 13: Comparação dos Resultados dos Testes de Precisão

É interessante observar que os modelos B e C tiveram resultados igualmente colineares com e sem aritmética exata. Em média, os erros de coplanaridade acumulados foram 55% menores quando se utilizou

Aritmética Exata. Em compensação, os erros de colinearidade, se mantiveram praticamente estáveis.

Desse fato, pode-se inferir que os pontos de interseção já ficavam posicionados de forma bem “colinear” em relação à suas arestas, mesmo sem Aritmética Exata. A isso, podemos atribuir o fato de utilizarmos a equação paramétrica da aresta/segmento para calcular o ponto de interseção, o que garante, naturalmente, a colinearidade do ponto de interseção.

Ao mesmo tempo, o posicionamento do ponto de interseção no triângulo sofreu uma pequena correção com a introdução da Aritmética Exata. Isso ocorreu porque o fator de interpolação, t_P, foi calculado de forma mais precisa, porém, sequer chegando a diminuir a ordem de grandeza dos erros dos pontos de interseção.

Assim, a grande contribuição da aritmética exata não foi o ganho de precisão dos pontos de interseção em si, e sim, o ganho de confiabilidade na classificação das interseções segmento-triângulo.

5.4. Testes de Eficiência

O tempo gasto pelo algoritmo proposto (unindo todas as etapas do algoritmo) será medido em três versões deste algoritmo:

- *Inexato*: sem Aritmética Exata, o que será feito substituindo os Predicados de Shewchuk por predicados equivalentes sem aritmética exata.
- *Exato*: com Aritmética Exata Não-Adaptativa em todo o algoritmo.
- *Adapt+Exato:* com Aritmética Exata Adaptativa na fase de descarte com o plano do triângulo e com Aritmetica Exata Não-Adaptativa nos cálculos posteriores.

Dessa forma, será possível medir, em duas diferentes modalidades, o impacto isolado da Aritmética Exata no algoritmo proposto. Isto é, sem contabilizar o *overhead* de tempo decorrente das alterações de código necessárias à introdução da Aritmética Exata em si.

Evidentemente, algoritmos que não utilizam Aritmética Exata não possuirão tais trechos de código e ainda permitem algumas otimizações particulares que não podem ser replicadas para algoritmos com

Aritmética Exata. Portanto, é de se esperar que estes algoritmos sejam mais eficientes do que o proposto, que, em contrapartida, é mais robusto.

Shewchuk(1996) demonstrou que os predicados de Aritmética Exata Adaptativos são ligeiramente mais lentos que os sem Aritmética Exata e que os predicados não-Adaptativos são bem mais lentos que os Adaptativos. Assim, é esperado que a versão sem Aritmética Exata do algoritmo proposto, que realiza bem menos operações aritméticas, seja a mais rápida. A estratégia adotada na terceira versão do algoritmo visa diminuir o custo da Aritmética Exata Não-Adaptativa, sabidamente mais lenta que a Adaptativa. Sendo assim, espera-se que a versão totalmente não-adaptativa seja a mais lenta das três.

Para efeito de comparação, foi fornecida a Tabela 14, contendo dados das malhas e das curvas geradas em cada modelo.

Modelo	Número de Triângulos	Número de Pares de Triângulos em Interseção	Número de Pontos de Interseção
Modelo A	1912 x 1912	252	127
Modelo B	2743 x 2743	160	81
Modelo C	2743 x 2743	732	367
Modelo D	1912 x 1912	528	268
Modelo E.1	265 x 858	108	55
Modelo E.2	265 x 890	114	58

Tabela 14: Dados das Malhas e Curvas de Interseção Geradas em Cada Modelo

Os tempos, cronometrados em milissegundos, estão expostos na Tabela 15, para cada modelo, versão do algoritmo e medição. As diferenças percentuais mostradas na Tabela 15 são relativas à média dos tempos da versão sem Aritmética Exata do algoritmo.

Tempos (ms)	Modelo A			Modelo B			Modelo C		
	Inexato	Adapt +Exato	Exato	Inexato	Adapt +Exato	Exato	Inexato	Adapt +Exato	Exato
1	1926	1973	2151	2623	2683	2757	2912	3060	3512
2	1916	1935	2152	2632	2677	2775	2927	2995	3492
3	1912	1933	2137	2643	2682	2773	2949	3048	3453
4	1936	1927	2135	2626	2684	2777	2912	3043	3556
5	1895	1936	2121	2657	2653	2798	2928	3049	3555
6	1918	1930	2164	2633	2646	2771	2924	3002	3327
7	1901	1926	2140	2629	2695	2764	2930	3015	3405
8	1894	1931	2135	2638	2688	2780	2924	3031	3168
9	1883	1929	2138	2629	2696	2777	2932	3027	3395
10	1922	1963	2136	2639	2668	2763	2914	3033	3013
Média	1910.3	1938.3	2140.9	2634.9	2677.2	2773.5	2925.2	3030.3	3387.6
Dif (%)	N/A	1.44%	10.77%	N/A	1.58%	5.00%	N/A	3.47%	13.65%

(a)

Tempos (ms)	Modelo D			Modelo E.1			Modelo E.2		
	Inexato	Adapt +Exato	Exato	Inexato	Adapt +Exato	Exato	Inexato	Adapt +Exato	Exato
1	1983	2062	2226	550	541	646	542	592	652
2	1950	2072	2333	531	563	650	557	564	652
3	1995	2158	2311	526	549	649	555	570	658
4	1978	2101	2504	531	545	651	548	562	655
5	1984	2149	2483	530	549	630	547	565	650
6	2005	2121	2031	529	548	649	543	587	648
7	2003	2114	2210	531	547	645	544	572	615
8	1950	2110	2325	527	544	654	540	570	647
9	1978	2102	2532	529	552	649	539	564	655
10	1973	2092	2513	531	583	646	545	572	650
Média	1979.9	2108.1	2346.8	531.5	552.1	646.9	546.0	571.8	648.2
Dif (%)	N/A	6.08%	15.63%	N/A	3.73%	17.84%	N/A	4.51%	15.77%

(b)

Tabela 15: Comparação dos Resultados dos Testes de Tempo

Como esperado, a versão sem Aritmética Exata foi mais rápida e a versão com Aritmética Exata somente Não-Adaptativa foi a mais lenta. Em particular na versão com Aritmética Exata Adaptativa e Não-Adaptativa, foi introduzido um custo adicional de, no mínimo, 1.44% e, no máximo, 6.08% no tempo total de processamento em relaçao a versão sem Aritmetica Exata. Na média dos exemplos mostrados, este gasto extra foi de 3.47%, o que é pouco, principalmente se comprado aos 13.11% da versão com Aritmética Exata somente Não-Adaptativa. Portanto, o uso de Aritmética Exata Adaptativa combinada com a Não-Adaptativa é capaz de, com uma leve perda de performance, aumentar a confiabilidade do algoritmo.

6

Conclusão

Neste trabalho, foi desenvolvido um algoritmo robusto para a interseção de superfícies triangulares representando superfícies geológicas. Nos testes realizados, este algoritmo se mostrou mais confiável, bastante versátil e relativamente eficiente. Apesar do ganho de confiabilidade, não houve um ganho de precisão significativo nos pontos de interseção com a estratégia adotada. Entre as principais contribuições deste trabalho estão:

- O algoritmo proposto utiliza uma R-Tree para busca de triângulos em interseção, permitindo grandes variações no tamanho dos triângulos, ao contrário do Algoritmo de LO & WANG (2004), que usam um grid regular. O método TNOIT também foi aprimorado, sendo capaz de construir múltiplos *loops* e *chains*, inclusive em superfícies com buracos (*genera*).
- Comprovou-se que a classificação da interseção segmento-triângulo é feita de forma mais confiável utilizando Aritmética Exata;
- O reaproveitamento do valor dos volumes com sinal avaliados com Aritmética Exata no cálculo do ponto de interseção não os torna significativamente mais precisos.
- A introdução de Aritmética Exata Adaptativa na classificação de interseções segmento-triângulo e da Aritmética Exata não-Adaptativa no cálculo dos pontos de interseção gerou um pequeno *overhead* de performance do algoritmo, apesar do grande ganho de confiabilidade.
- A ordenação lexicográfica permitiu que o cálculo de Interseção Segmento-Triângulo fosse realizado de forma independente da ordem dos pontos do segmento e do

triângulo. Isso garante que a interseção da superfície S1 sobre S2 será a mesma de S2 sobre S1.

Algumas limitações deste trabalho são:

- Não trata interseção de “superfícies em contato”, isto é, quando a borda de uma superfície está sobre outra superfície. No algoritmo implementado, é preciso que haja triângulos cortando outros triângulos e não apenas tocando-se.
- As superfícies a serem intersectadas não podem possuir triângulos coplanares em interseção.

Como trabalhos futuros, pode-se citar:

- Implementar operação de divisão com aritmética exata, intervalar ou afim;
- Comparar mesmo algoritmo usando aritmética afim e intervalar;
- Tentar minimizar as chamadas recursivas ou substituí-las por operações em pilha, que é uma estratégia sabidamente mais eficiente;
- Recriar o algoritmo proposto utilizando Aritmética Exata diretamente, ao invés de por meio dos predicados, o que tornaria o algoritmo mais rápido;
- Regerar a triangulação das superfícies ao mesmo tempo em que se constrói a curva, realizando modificações locais na malha.

Referências

AFTOSMIS, M. J., BERGER, M. J., MELTON, J. E , **Robust and Efficient Cartesian mesh generation for component-based geometry**, *35th AIAA Aerospace Sciences Meeting and Exhibit, 6-9 January 1997.*

ALVES, A.L.F., SILVA, R.M., **Geração da Curva Interseção de Superfícies Trianguladas**. *Workshop of Undergraduate Work, SIBGRAPI, 2008*

ARRUDA, M.C., MARTHA, L.F. Operações Booleanas com Sólidos Compostos Representados por Fronteira, *Dissertação de Mestrado, PUC-Rio, 2005*

BAUMGART, B. G., **Winged edge polyhedron representation.**, Relatório Ténico. Universidade de Stanford, Califórnia, Estados Unidos, 1972

BECKMANN, N., KRIEGEL, H.P., SCHNEIDER, R., SEEGER, B. **The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles.** *Proceedings of the ACM SIGMOD Conference on Management of Data, pp 322-332, Maio 1990*

CASS, R.J., BENZLEY, S.E., MEYERS, R.J., BLACKER, T.D. **Generalized 3-D paving: an automated quadrilateral surface mesh generation algorithm**. *Int J Num Meth Eng, Volume 39(9), pages 1475–1489, 1996*

COELHO, L. C., GATTASS, M., FIGUEREDO, L. H. **Intersecting and trimming parametric meshes on finite element shells**. *International Journal for Numerical Methods in Engineering, vol.0(0),1-10, 1999.*

CUILLIÈRE, J.C., **An adaptive method for the automatic triangulation of 3D parametric surfaces**, *Comput.-Aid. Des. Volume 30(2), pages 95–161, 1998*

FARIN G., **Curves and Surfaces for Computer Aided Geometric Design**, *Academic Press, 1990*

FIGUEIREDO, L.H., STOLFI, J. **Affine Arithmetic: Concepts and Applications,** *Numerical Algorithms, Vol 37, pp 147-158, 2004*

FORTUNE, S. WYK, C.J.V., **Efficient Exact Arithmetic for Computational Geometry**, *Proceedings of the Ninth Annual Symposium on Computational Geometry, pp. 163-172, Association for Computing Machinery, 1993*.

GUTTMAN, A., **R-Trees: A Dynamic Index Structure for Spatial Searching**. *Proceedings of ACM SIGMOD International Conference on Management of Data, pp 47, 1984*

HOFFMANN, C.M., **Geometric and Solid Modelling**, *2nd edition, 1992*.

JIMÉNEZ, J.J., SEGURA, R.J., FEITO, F.R., **A Robust Segment-Triangle Intersection Algorithm for Interference Tests. Efficiency Study**. *Computational Geometry Volume 43, Issue 5, Pages474-492,* 2009

KETTNER, L., MEHLHORN K., PION, S., SCHIRRA, S., YAP, C., **Classroom Examples of Robustness Problems in Geometric Computations,** *Computational Geometry, Volume 40, pages 61-78, 2008*

LAGE, M., LEWINER, T., LOPES, H., VELHO, L., **CHE: A Scalable Topological Data Structure for Triangular Meshes**, *SIBGRAPI, pages 349-356, 2005*

LAMAS, G.N., LOPES, H.C.V. **Aritmética intervalar: implementação e algumas aplicações.** *Dissertação (Mestrado)-Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Matemática, 1999*

LAUG P, BOROUCHAKI H. **Molecular surface modeling and meshing**. *Proceedings of the 10th International Meshing Roundtable, Newport Beach, CA, 7–10 October 2001*

LO, S.H. **Automatic mesh generation over intersecting surfaces**, *International Journal for Numerical Methods in Engineering,Volume 38, pages 943–954, 1995.*

LO, S.H., WANG, W.X. **A Fast Robust Algorithm for the Intersection of Triangulated Surfaces**. *Engineering with Computers, 2004.*

LO, S.H., WANG, W.X. **Finite element mesh generation over intersecting curved surfaces by tracing of neighbours**. *Finite Elements in Analysis and Design 41, pages 351–370, 2005*

LÖHNER, R. **Extending the range of applicability and automation of the advancing front grid generation technique**, *34th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1996.*

LYRA, W.W.M., **Um Sistema Inetgrado Configurável para Simulações em Mecânica Computacional.** *Dissertação de Mestrado, Departamento de Engenharia Civil, PUC-Rio, Rio de Janeiro, 1998.*

MALLET J., **Discrete Smooth Interpolation**, *ACM Transaction on Graphics, Vol 8, No 2, Pages 121-144, April 1989*

MÄNTYLÄ, M. **An Introduction to Solid Modeling.** *Computer Science Press, Rockville, MD, 1988*

MARQUES, R.C., PEREIRA, A., MARTHA, L.F., MIRANDA, A., GATTASS, M., **Adaptive Precision Based Fast Algorithm for Robust Surface Intersections**, *Mecom/Cilamce*, 2010

MARTINETZ, T., SCHULTEN, K.**A Neural Gas Network Learns Topologies,** *Artificial Neural Networks, 1991*

MEAGER, D. **Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3D Objects by Computer**. *Relatório Técnico IPL-TR-80-111. Rensselaer Polytechnic Institute. 1980.*

MIRANDA, A.C.O., MARTHA, L.F., WAWRZYNEK, P.A., INGRAFFEA, A.R. **Surface Mesh Generation Considering Curvatures**. *Engineering with Computers, Vol 25, Issue 2,, pages 207-219, Mar2009*

MÖLLER, T.A., **A Fast 3D Triangle-Triangle Intersection Test**, *Journal of Graphics Tools, Vol2, Issue 2, pages 25-30, 1997*

MÖLLER, T.A., **Fast 3D Triangle-Box Overlap Testing**, *Journal of Graphics Tools, Vol 6, Issue 1, pages 29-33, Set/2001*

MÖLLER, T., TRUMBORE, B, **Fast, Minimum Ray/Triangle Intersection**, *Journal of Graphics Tools, Vol2, Issue 1, pages 21-28, 1997.*

MOORE, R.E. **Methods and Applications of Interval Analysis.** *Vol. 2. Philadelphia: Siam, 1979.*

O'ROURKE, J., **Computational Geometry in C.** *Cambridge, Univ. Press, 1994.*

PEREIRA, A.M.B., ARRUDA, M.C., MIRANDA, A.C.O, LIRA, W.W.M., MARTHA, L.F., **Boolean Operations on Multi-Region Solids for Mesh Generation,** *Engineering with Computers, Vol 28, pages 225-239, 2012*

PRIEST, D.M., **Algorithms for Arbitrary Precision Floating Point Arithmetic**, *Tenth Symposium on Computer Arithmetic, pp. 132-143, IEEE Computer Society Press, 1991*

SANTOS, J. A. ARAÚJO, C. E. S. e SILVA, R. M.. **Geração de um Modelo Geométrico de Canais Encontrados em Reservatórios Petrolíferos**. *XVIII Brazilian Symposium on Computer Graphics and Image Processing, SIBGRAPI - WORKSHOP of Undegraduate Work, Natal, RN, 2005.*

SEGURA, R.J., FEITO, F.R., **Algorithms to Test Ray-Triangle Intersection.** Comparative Study. *Journal of WSCG*, 2001

SEGURA, R.J., FEITO, F.R., **An Algorithm for Determining Intersection Segment-Polygon in 3D.** *Computer & Graphics*. 1998

SELLIS, T, ROUSSOPOULOS, N, FALOUTSOS, C. **The R+Tree: A Dynamic Index for Multi-Dimensional Objects**. *International Conference on Very Large Data Bases (VLDB), 1987*

SHEPHARD, M.S., GEORGES, M.K. **Automatic 3-dimensional mesh generation by the finite octree technique.** *International Journal for Numerical Methods in Engineering, Volume 32(4), pages 709–749, 1991*

SHEWCHUK, J.R., **Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates**, *Discrete and Computational Geometry, Vol 18, No 3, pages 305-368, 1996a*

SHEWCHUK, J.R., **Robust Adaptive Floating-Point Geometric Predicates**, *Twelfth Annual Symposium on Computational Geometry*, pages 141-150, May 1996b

SHOSTKO, A.A., LÖHNER, R., SANDBERG, W.C. **Surface triangulation over intersecting geometries**. *Int J Num Meth Eng, Volume 44, pages 1359–1376, 1999*

WEATHERHILL, N.P. **Delaunay triangulation in computational fluid dynamics**, *Comput. Math. Appl., 24(5/6), 129-150, 1992.*

WEILER, K. **The radial edge structure: a topological representation for non-manifold geometric boundary modeling**. *Geometric Modeling for CAD Applications, pages 3-36, 1988*