UNIVERSIDADE FEDERAL FLUMINENSE

ALTOBELLI DE BRITO MANTUAN

Um método para identificar linhas de fratura em fragmentos tridimensionais.

NITERÓI

2011

UNIVERSIDADE FEDERAL FLUMINENSE

ALTOBELLI DE BRITO MANTUAN

Um método para identificar linhas de fratura em fragmentos tridimensionais.

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel em Ciência da Computação do curso de Ciência da Computação da Universidade Federal Fluminense.

Orientador:
Helena Cristina da Gama Leitão

NITERÓI

2011

Um método para identificar linhas de fratura em fragmentos tridimensionais.

Altobelli de Brito Mantuan

Trabalho de conclusão de curso apresentado como parte das atividades para obtenção do título de Bacharel em Ciência da Computação do curso de Ciência da Computação da Universidade Federal Fluminense.

Banca Examinadora:

Prof. Dante Corbucci Filho / IC-UFF

Prof. Marcos de Oliveira Lage Ferreira / IC-UFF

Profa. Helena Cristina da Gama Leitão / IC-UFF
(Orientadora)

Niterói, 18 de Agosto de 2011.

"Como dois e dois são quatro
Sei que a vida vale a pena
Embora o pão seja caro
E a liberdade pequena.."
Ferreira Gullar

Agradecimentos

Considerando esta monográfia como resultado de uma caminhada que não começou na UFF, agradecer pode não ser tarefa fácil, nem justa. Para não correr o risco da injustiça, agradeço de antemão a todos que de alguma forma passaram pela minha vida e contribuíram para a construção de quem sou hoje.

E agradeço, particularmente, a algumas pessoas pela contribuição direta na construção deste trabalho:

Primeiramente à meus pais, Antônio e Marlene, pelo apoio incondicional. Se há alguém responsável por tudo o que sou hoje são vocês. Obrigado pelo amor, boa educação, criação, sabedoria e, acima de tudo, pela amizade.

À professora Helena Cristina da Gama Leitão, pela discussão teórica sobre o projeto, por ter sido atenciosa e dedicada na orientação desta monográfia.

Os professores Jorge Stolfi e Rafael Felipe Veiga Saracchini, pela orientação para com o tema do trabalho.

Aos meus amigos Ana Paula, Helena, Alexandre, Igor, Ayrton, as meninas do 302, Henrique e Estevão por fazerem de Niterói minha segunda casa.

Resumo

Neste trabalho, apresentamos um método para determinar as linhas de fronteiras de um fragmento tridimensional. Para esse fim realizamos um estudo sobre: o comportamento da representação vetorial destes fragmentos, chamada de normais; as propriedades dos invariantes geométricos, especificamente a curvatura e torção, e por último, sobre o uso da estrutura *Half-edge*, importante para a manipulação dos dados do objeto.

Abstract

In this work, we present a method that determine borders lines from three-dimensional fragment. For this purpose, we research for: the vector representation's behavior, called normal; the properties of geometric invariants specifically the curvature; torsion and finally about the usage of half-edge structure, relevant for handling the object's data.

Palavras-chave

1. Half-edge
2. Reconstrução Tridimensional
3. Linhas de borda
4. Invariantes geométricos
5. Variação das normais

Conteúdo

Capítulo 1

Introdução

A modelagem, comparação e identificação de objetos tridimensionais (3D) são importantes em áreas como arqueologia, medicina e outros, onde o objeto a ser analisado, pode ser danificado, ou até mesmo destruído devido a fragilidade do mesmo, se manipulado diretamente. Para ajudar a resolver esse problema buscam-se técnicas computacionais e equipamentos para modelagem de tais objetos que permitam sua manipulação e análise, sem riscos para o objeto real.

Umas das formas de se obter a descrição 3D de um objeto é através do equipamento scanner 3D. Porém nem sempre temos disponível tal equipamento e existem outras formas de obtenção da representação 3D do objeto. Um dos métodos para obtenção dessa representação é a estereofotométria, que calcula a altura de um objeto a partir de três ou mais imagens, tiradas do mesmo ponto de vista, mas sob iluminação diferentes.

Neste projeto, utilizamos um programa estereofotométrico desenvolvido por Saracchini e Morais[1], para obtenção dos dados 3D dos fragmentos em estudo. Com os dados tridimensionais do objeto, apresentaremos técnicas para identificação de linhas de fratura, ou seja, pontos no fragmento que são considerados bordas. Estas informações de bordas são importantes para achar pares de fragmentos adjacentes do objeto original.

1.1 Motivação

O projeto de Reconstrução de Artefatos Arqueológico Brasileiros (RAAB)[2], onde nasceu essa ideia, surgiu da necessidade dos pesquisadores do Instituto Arqueológico Brasileiro(IAB) em trabalhar com representações tridimensionais dos objetos reais.

O projeto RAAB reuniu alguns trabalhos que vão desde a reconstrução da represen-

tação gráfica do objeto até a identificação de trechos adjacentes. Na área de identificação de pares de fragmentos, foi feito um trabalho por Leitão, Stolfi[3] usando processamento de imagens, na maioria dos casos os testes foram satisfatórios, para fragmentos planares. A necessidade de um novo estudo para tratar objetos não planos, motivou o atual trabalho.

1.2 Trabalhos na área

Vários dos trabalhos encontrados na área de reconstrução são motivados pela remontagem de artefatos arqueológicos. O problema de identificação de objetos adjacentes utilizando linhas de contorno foi considerado por Üçoluk, Toroslu[4], Martin, Sablating[5] e Weixin, Benjamin[6], mostrando técnicas de comparação de objetos 2D, utilizando como referência o contorno da imagem do objeto, e demonstrando o uso de invariantes geométricos para comparação entre fragmentos.

Leitão e Stolfi[3] apresentam o uso da técnica de escalas múltiplas, para reduzir a amostra de pares de fragmentos que têm mais possibilidade de serem adjacentes no objeto original. A ideia consiste em representar os elementos em precisão menores e fazer a comparação dos mesmos, neste momento, já são eliminados fragmentos que não possuem trechos similares. Em seguida, aumenta-se a precisão dos elementos e compara-se os pares, assim sucessivamente.

São encontrados também na literatura trabalhos sobre reconstrução com elementos 3D, neste caso é abordado o uso de outros invariantes geométricos, para a comparação entre fragmentos tridimensionais. Podemos citar Tsinghua, Vienna, Stanford[7] que apresenta um nova técnica para calcular invariante e trechos adjacentes de forma global e local. Podemos ver nos trabalhos kampel[5], Toroslu[4] e Willis, Cooper[8], o problema de obter a linha de borda do elemento tridimensional, e o uso da técnica de regressão linear para obter melhor a representação da curva de contorno.

Capítulo 2

Conceitos Básicos

2.1 Geometria Espacial

Os vetores são instrumentos úteis no estudo da geometria plana, e são quase indispensáveis na geometria espacial. Eles são definidos como classes de equipolência de segmentos orientados, ou seja, os segmentos de reta orientados, são equipolentes quando cumprem as seguintes condições: têm o mesmo comprimento; são paralelos ou colineares; têm o mesmo sentido [9].

A seguir, serão mostrados alguns conceitos, propriedades e aplicações sobre vetores, úteis para este projeto.

2.1.1 Soma de Vetores

Dado os vetores $v = (v_1, v_2, v_3)$ e $w = (w_1, w_2, w_3)$, definimos a soma de v e w, por:

$$v + w = (v_1 + w_1, v_2 + w_2, v_3 + w_3) \tag{2.1}$$

A seguir, mostramos as propriedades da adição de vetores. Eles são:

- Fecho: Para quaisquer u e v de R^3, a soma $u + v$ está em R^3.
- Comutativa: Para todos os vetores u e v de R^3: $v + w = w + v$.
- Associativa: Para todos os vetores u, v e w de R^3: $u + (v + w) = (u + v) + w$.
- Elemento neutro: Existe um vetor $\emptyset = (0, 0, 0)$ em R^3 tal que para todo vetor u de R^3, se tem: $\emptyset + u = u$.

- Elemento oposto: Para cada vetor v de R^3, existe um vetor $-v$ em R^3 tal que: $v + (-v) = \emptyset$.

2.1.2 Produto Escalar

O produto escalar ou interno de dois vetores é um número, a partir do qual se pode obter informações de distância e ângulo [10]. Seja, portanto, os vetores $v = (v_1, v_2, v_3)$ e $w = (w_1, w_2, w_3)$, definimos o produto escalar entre v e w, como o escalar real:

$$v.w = v_1w_1 + v_2w_2 + v_3w_3 \tag{2.2}$$

Quaisquer que sejam os vetores u, v, w e o escalar k:

- (PE1) $v.w = w.v$;
- (PE2) $v.v = |v||v| = |v|^2$;
- (PE3) $u.(v + w) = u.v + u.w$;
- (PE4) $(kv).w = v.(kw) = k(v.w)$;
- (PE5) $|kv| = |k||v|$;
- (PE6) $|u.v| < |u|.|v|$ (desigualdade de Schwarz [11]);
- (PE7) $|u + v| < |u| + |v|$ (desigualdade triangular [12]) .

2.1.3 Produto Vetorial

O produto vetorial é um vetor, resultante da combinação de dois vetores no espaço [10]. Sendo os vetores $v = (v_1, v_2, v_3)$ e $w = (w_1, w_2, w_3)$, definimos o produto vetorial entre v e w, denotado por $v \times w$, como o vetor ortogonal a v e também a w, isto é, o produto vetorial é ortogonal ao plano que contém os dois vetores v e w.

$$v \times w = \begin{pmatrix} i & j & k \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{pmatrix} \tag{2.3}$$

Propriedades do Produto Vetorial, dado os vetores u, v e w qualquer e um escalar k:

- (PV1) $v \times w = -w \times v$;
- (PV2) $u \times (v + w) = u \times v + u \times w$;
- (PV3) $k(v \times w) = (kv) \times w = v \times (kw)$;
- (PV4) $u \times u = v \times v = w \times w = 0$;
- (PV5) $u \times v = w, v \times w = u, w \times u = v$;
- (PV6) Se $v \times w = 0$, se somente se, v e w são colineares.

2.1.4 Aplicações Geométricas

Dado $v = (x_1, y_1, z_1)$, $u = (x_2, y_2, z_2)$ e $w = (x_3, y_3, z_3)$, segue os seguintes conceitos:

Ponto médio de um segmento: Dado um segmento de reta, cujo os pontos são v e u, o ponto médio deste segmento é dado por $m = (x, y, z)$ onde

$x = \frac{(x1+x2)}{2}$; $y = \frac{(y1+y2)}{2}$; $z = \frac{(z1+z2)}{2}$

Centro de massa de um triângulo: Consideremos os vértices de um triângulo, composto por v, u e w. O centro de gravidade deste triângulo é dado pelo vetor $g = (x, y, z)$ onde:

$x = \frac{(x_1+x_2+x_3)}{3}$; $y = \frac{(y_1+y_2+y_3)}{3}$; $z = \frac{(z_1+z_2+z_3)}{3}$

Diferença de vetores: Dado os vetores v e u, definimos a diferença entre v e u, por:

$v - u = (x_1 - x_2, y_1 - y_2, z_1 - z_2)$

Módulo de vetor: O módulo ou comprimento do vetor $v = (x, y, z)$ é definido por:

$|u| = \sqrt{x^2 + y^2 + z^2}$

Vetor Direção: Dado um segmento de reta, definido pelos vértices v e u, para achar o vetor direção $d = (x, y, z)$, sendo w o vetor diferença entre $u - v$; temos:

$d = \frac{w}{|w|}$

2.2 Mapa de Normais

Uma normal é um vetor perpendicular ao plano. Um mapa de normais contém as informações dos vetores normais a cada face ou vértice que constitui o objeto. Elas são muito

importantes na iluminação de objetos tridimensionais, e podem ser usadas para descrever a orientação de uma superfície.

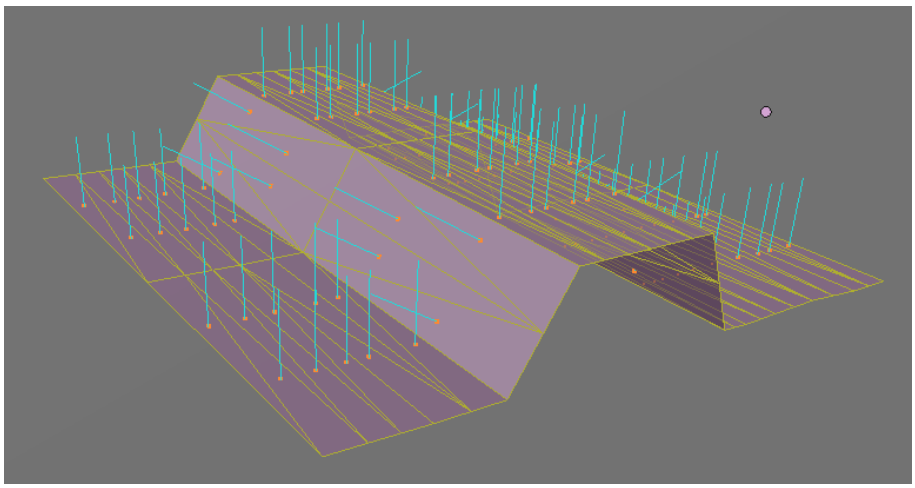

Figura 2.1: Exemplo de normal a superfície.

Além disso, também pode ser definida a normal de vértice, para isto é necessário somar todas as normais das faces que compartilham esse vértice, e dividir pela quantidade de faces; sendo n número de faces, $v_i = (x_i, y_i, z_i)$ vetor normal de cada face que compartilha o vértice a ser calculado a normal e $n_j = (x, y, z)$ vetor normal do vértice, segue a fórmula:

$$n_j = \frac{\sum_{i=0}^{n}(v_i)}{n} \tag{2.4}$$

2.3 Geometria Diferencial

2.3.1 Invariantes Geométricos

Invariantes geométricos são um ou mais atributos que não sofrem alterações, mesmo se o objeto for transladado ou rotacionado. Essa qualidade dos invariantes, também chamado de assinatura do objeto por Mundy e Zisserman[13], é tipicamente usada quando é necessário reconhecer aspectos de um objetos para sua identificação; por isso são amplamente utilizados em visão computacional. Os invariantes são classificados em dois tipos: global, que tenta extrair características como um todo do objeto de estudo, e local, que segmenta em partes menores para extrair informações. Temos no exemplo a seguir, a figura 2.2(a), curva original, e temos a abaixo na figura 2.2(b), a mesma curva que sofreu uma rotação. Porém fica visível notar que olhando a curva em diferente ponto de vista ela muda sua aparência, ficando difícil afirmar se elas são realmente idênticas.

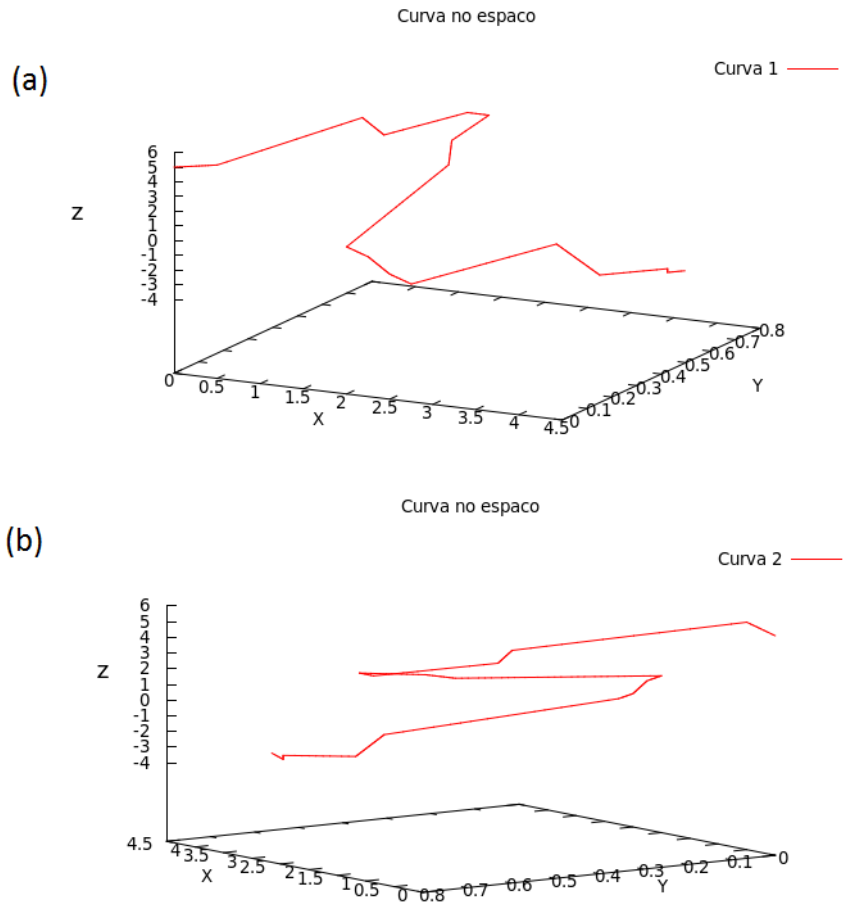

Figura 2.2: Exemplos de curvas no espaço: (a) curva original; (b) curva rotacionada.

Neste trabalho vamos falar somente de dois tipos de invariantes geométricos: curvatura e torção. É importante lembrar que existem outros tipos de invariantes, porém essas assinaturas dependem do objeto a ser trabalhado. A seguir, vamos demonstrar como calcular invariantes locais dessas curvas.

2.3.1.1 Cálculo de Curvatura

O processo para o cálculo de curvatura Craizer[14], Weiss[15], utiliza três pontos consecutivos de uma curva, dado os pontos Pq, Po, Pq_1, sendo Po o ponto central entre Pq e Pq_1.

Através desses dados, vamos achar o ângulo entre os vetores $u(Po, Pq)$ e $v(Po, Pq_1)$, para obtermos uma estimativa da curva no ponto Po, temos que calcular o ângulo entre os vetores $\measuredangle(u, v)$, sendo $k(Po)$, o cosseno do ponto Po, dado por:

$$k(Po) = \frac{u \cdot v}{|u| \cdot |v|} \tag{2.5}$$

Onde $k \subset [-1, 1]$, visto o procedimento para identificar curvatura, vamos usar esse procedimento sobre a curva da figura 2.2(a), vamos mostrar que a curvatura dos pontos da curva da figura 2.2(b) não se alteram. Na figura 2.3, temos os resultados cálculados para cada ponto das curvas.

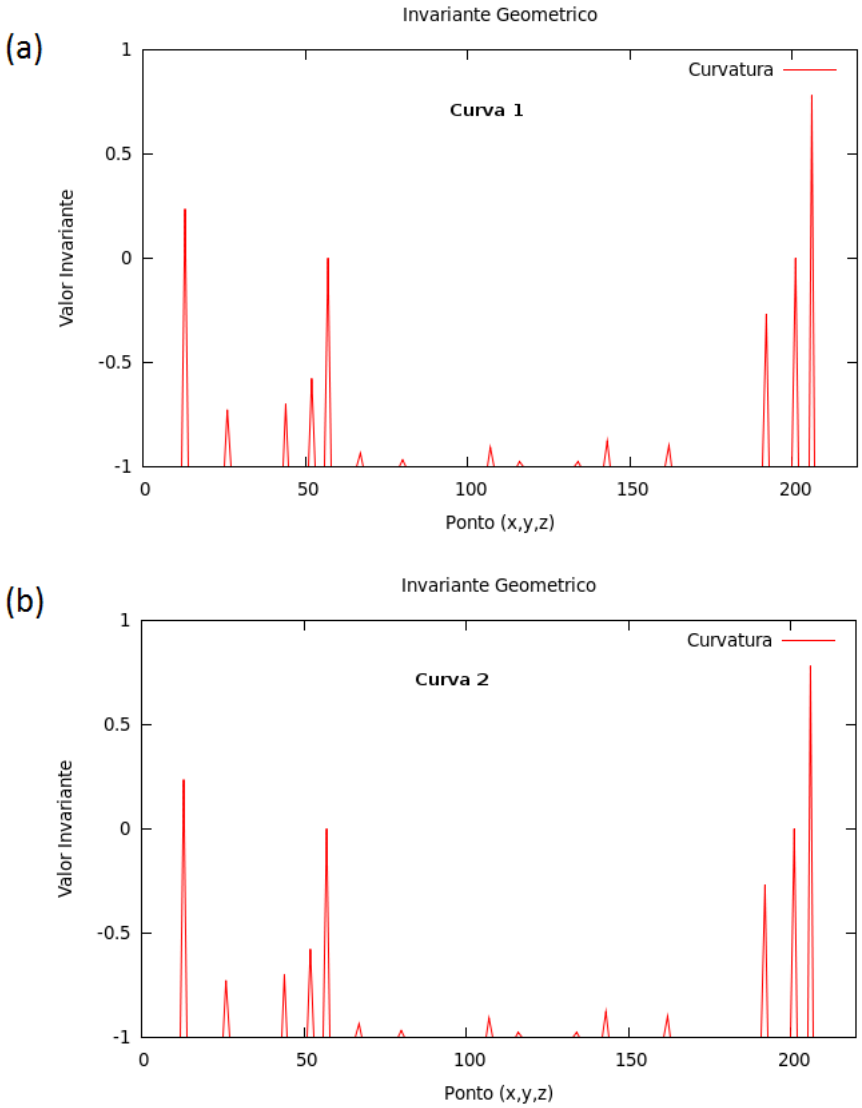

Figura 2.3: Resultados dos cálculos de curvatura:(a)valores da curvatura calculada sobre a curva 1; (b)valores da curvatura calculada sobre a curva 2.

2.3.1.2 Cálculo de Torção

A torção ð mede, em cada ponto o quão a curva se afasta do plano; para o cálculo da torção [14] são necessários 4 pontos consecutivos, vamos chamar de Pq,Po,Pq_1 e P_t. Dado os vetores $u(Po, Pq)$ e $v(Po, Pq_1)$, podemos achar o vetor ortogonal através do produto vetorial $u \times v$, esse vetor é ortogonal a v e também ortogonal a u, e ortogonal ao plano que contém os dois vetores u e v; temos outro vetor $z(Pq_1, P_t)$, necessário para o cálculo da torção. Agora podemos calcular $ð(P_t)$, dado por:

$$ð(P_t) = \frac{(u \times v) \cdot z}{|u \times v| \cdot |z|} \tag{2.6}$$

Onde $\eth(P_t) \subset [-1, 1]$; a seguir, na figura 2.4, mostramos os resultados do cálculo da torção nas curvas, que podem ser vistas na figura 2.2.

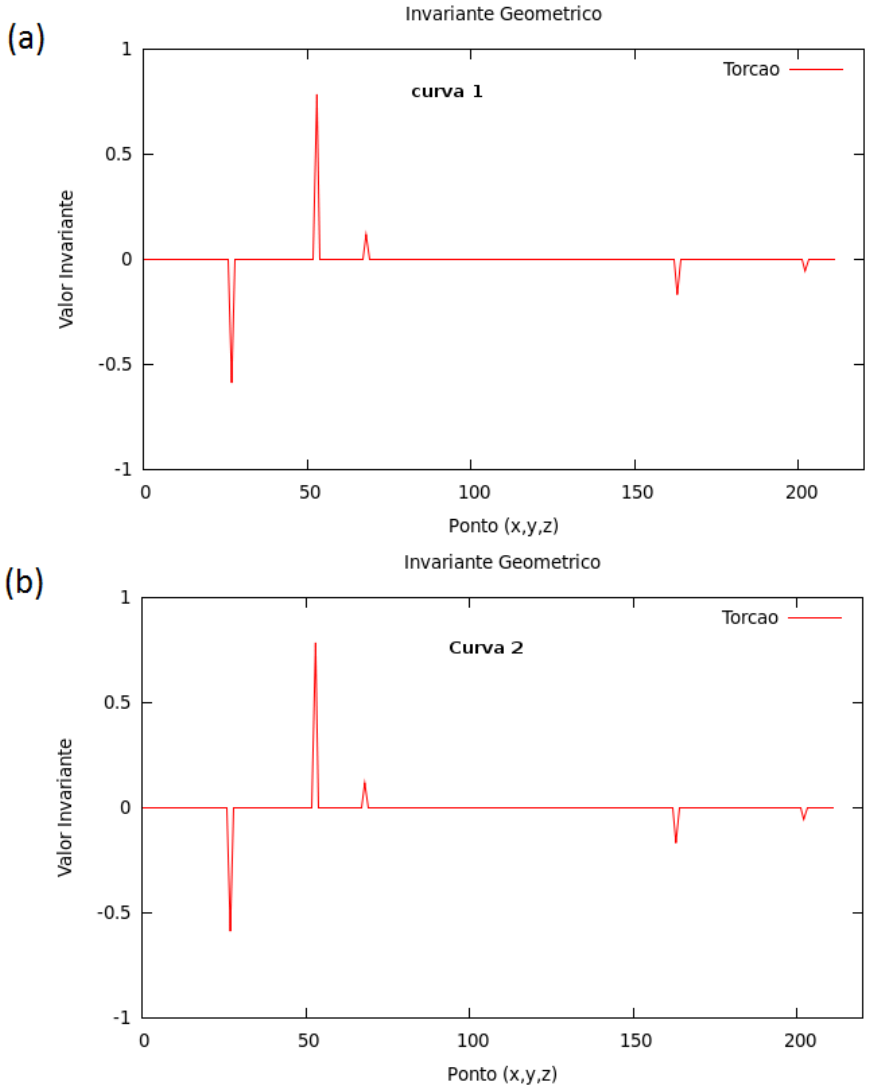

Figura 2.4: Resultados dos cálculos da torção: (a)valores da torção calculada sobre a curva 1; (b)valores da torção calculada sobre a curva 2

2.4 Objeto Gráfico Espacial

Um objeto gráfico espacial é um objeto que está no espaço de dimensão três, ou seja, as curvas espaciais(objetos 1D), as superfícies(objetos 2D), sólidos e objetos volumétricos fazem parte dessa definição.

Neste projeto, trabalhamos com superfícies poliédricas, pois estamos interessados nas informações da superfície do objeto.

2.4.1 Modelo Poliédrico

2.4.1.1 Superfície Poliédrica

Superfície poliédrica é um conjunto finito de polígonos planos cuja disposição no espaço satisfaz as seguintes propriedades [9]:

- P1. Todo polígono da superfície poliédrica possui algum lado em comum com outro polígono da superfície poliédrica, ou a superfície poliédrica é formada por um único polígono;
- P2. Todo lado de um polígono da superfície poliédrica pertencem à no máximo dois polígonos;
- P3. Superfície poliédrica aberta possui arestas que pertence a uma única face, caso contrário é fechada.

Os principais elementos em uma superfície poliédrica, como ilustra a figura 2.5, são:

- Faces: são polígonos da superfície poliédrica.
- Arestas: São os lados dos polígonos.
- Vértices: são os pontos dos polígonos.

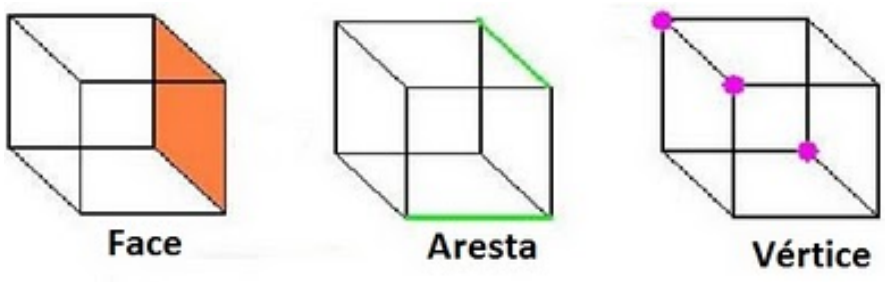

Figura 2.5: Elementos da superfície poliédrica.

Capítulo 3

Metodologia

3.1 Visão Geral

Na literatura vemos que o processo de reconstrução de objetos envolve algumas etapas que vão desde a obtenção de dados da representação do objeto real, até a remontagem do objeto original. Neste projeto, trabalhamos com objetos reais, obtidos a partir da técnica estereofotométrica, utilizando um programa desenvolvido por Saracchini e Morais[1], e objetos virtuais gerados por modelador gráfico.

Os dados dos fragmentos tridimensionais são definidos por faces, arestas e vértices. Combinamos essas informações em uma estrutura de dados *Half-edge*, para facilitar a manipulação do fragmento. Definido os dados e a estrutura a ser utilizada, o próximo passo é obter informações que são importantes para a comparação de fragmentos. A identificação da borda de um objeto é o aspecto mais importante na comparação entre fragmentos, uma abordagem para obter essas bordas é verificando o comportamento da normal sobre a superfície do objeto.

3.2 Cálculo da Variação das Normais

Para calcular a variação das normais apresentamos dois métodos: o primeiro que trabalha diretamente com o mapa de normais e o segundo utiliza superfície poliédrica.

3.2.1 Primeiro método

Seguindo a abordagem proposta Mantuan e Leitão[16] utilizamos o mapa de normais abaixo, como ilusta a figura 3.1, obtido a partir do programa estereofotométrico.

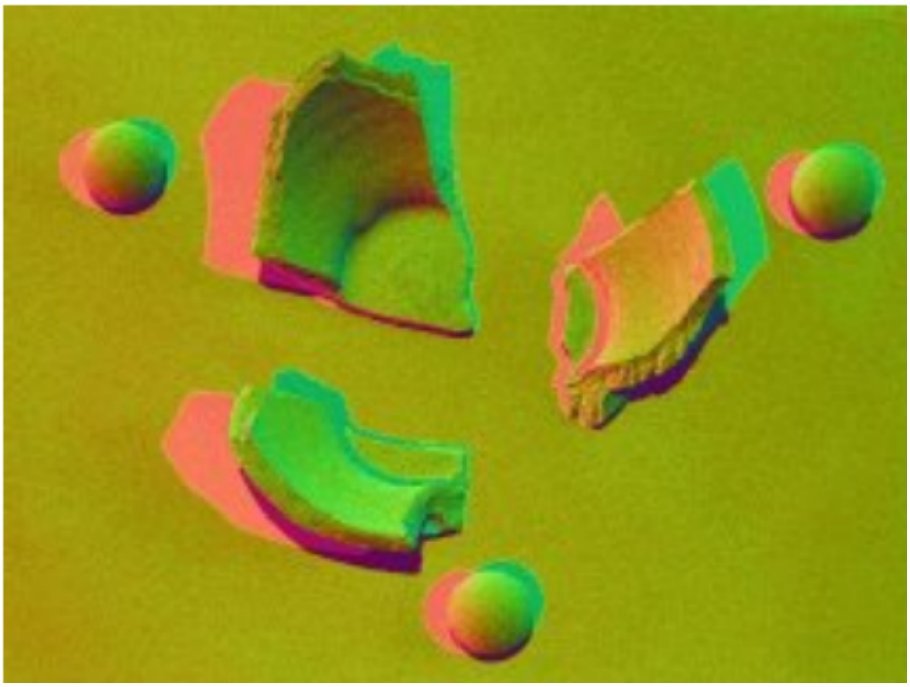

Figura 3.1: Exemplo de mapa de normais gerado pelo programa estereofotométrico.

Para os cálculos são utilizados a matriz $M_{n\times m}$ que representa o mapa de normais, o parâmetro r, que define a vizinhança ao um ponto central, que é utilizada para selecionar as normais no perímetro definido por r. Definimos também, $V = v_1, v_2, ..., v_k$ um conjunto de vetores normais da vizinhança, que é utilizado da seguinte forma; no primeiro elemento v_1 guardamos o vetor normal referente ao ponto central $M_{x,y}$, sendo $x \in 0, .., n-1$ e $y \in 0, .., m-1$, e nas seguintes, $\forall d \in 2, ..., k$ de v_d, estão os vetores normais dos pontos vizinhos selecionados. Segue abaixo o algoritmo:

Algoritmo 1: ObterPontosVizinhos

Entrada: $M_{n\times m}$, $M_{x,y}$, r
Saída: V
início
  $i \in 0, .., n-1$;
  $j \in 0, .., m-1$;
  para $\forall M_{i,j} \in M_{n\times m}$ **faça**
    se $M_{i,j} \subset$ *perímetro definido por r em relação a* $M_{x,y}$ **então**
      adiciona o vetor normal do ponto $M_{i,j}$ no conjunto V;
    fim
  fim
fim

Calculamos então, para cada ponto do mapa de normais, a matriz $W_{3\times2}$ que será utilizada para encontrar o valor de Q, sendo este, o valor da variação da normal no ponto $M_{x,y}$ em relação aos seus vizinhos.

$$Tx = \frac{1}{(v_{[i]}.x - v_{[1]}.x)} \tag{3.1}$$

$$Ty = \frac{1}{(v_{[i]}.y - v_{[1]}.y)} \tag{3.2}$$

$$W = \begin{bmatrix} \sum_{i=2}^{N}(Tx * (v_{[i]}.x.Nx)) & \sum_{i=2}^{N}(Ty * (v_{[i]}.y.Nx)) \\ \sum_{i=2}^{N}(Tx * (v_{[i]}.x.Ny)) & \sum_{i=2}^{N}(Ty * (v_{[i]}.y.Ny)) \\ \sum_{i=2}^{N}(Tx * (v_{[i]}.x.Nz)) & \sum_{i=2}^{N}(Ty * (v_{[i]}.y.Nz)) \end{bmatrix} \tag{3.3}$$

$$Q = \sqrt{\sum_{i=1}^{3}\sum_{j=1}^{2}(W_{ij})^2} \tag{3.4}$$

Este cálculo segue ao longo de todos os pontos do mapa de normal, tendo no final, para cada ponto uma variação associada. Segue abaixo, na figura 3.2 o resultado, os pontos escuros definem as maiores variações das normais.

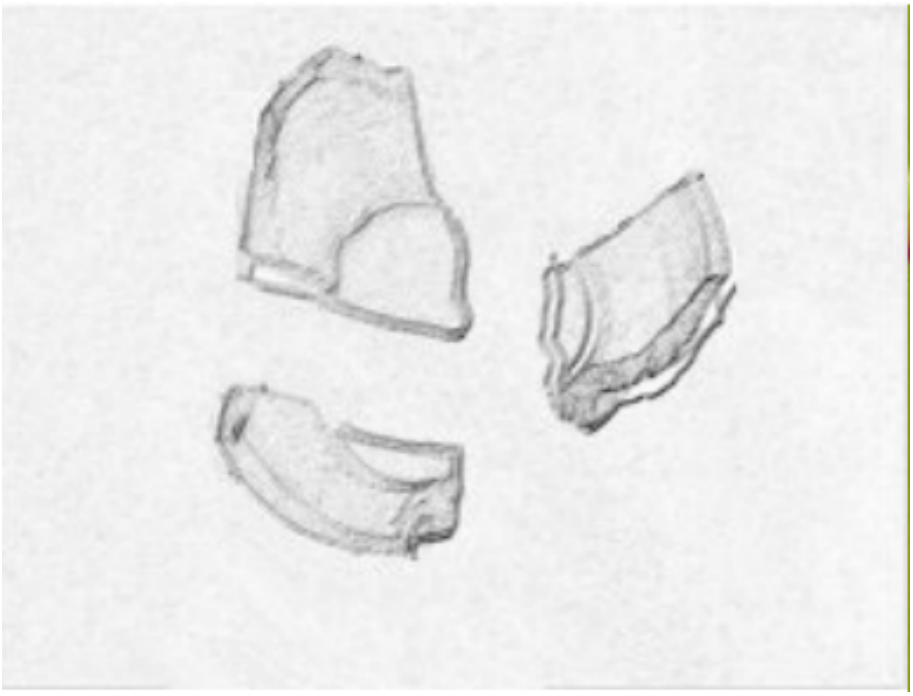

Figura 3.2: Saída do método. Os pontos escuros definem locais de maiores variações das normais.

3.2.2 Segundo Método

A segunda abordagem segue a mesma ideia sobre comparar a normal com a vizinhança, porém neste momento estamos trabalhando num modelo poliédrico composto por: faces, arestas e vértices. Para o cálculo da variação de normal, será usado o vetor normal a cada vértice.

Para obtenção dos vértices vizinhos ao vértice central v_j, é definido o raio r, que significa o alcance da vizinhança a v_j, que é utilizada para selecionar as normais neste perímetro. Definimos também, $C_1, C_2, ..., C_r$, que é um conjuntos de vetores de normais, que é definido da seguinte forma; no conjunto C_1 guardamos os vetores normais referente aos vértices que estão a uma unidade de distância do vértice central v_j, e nos seguintes, sendo $d \in 1, .., r$ de C_d, sendo $C_d = n_1, n_2, ..., n_m$, que é um conjunto de normais dos vértices selecionados, dado a unidade de distância a v_j, sendo definido $n_h \in C_d$. Segue abaixo o algoritmo:

Algoritmo 2: ObterVérticeVizinhos

Entrada: v_j, r
Saída: C_d
início
  se $r == 0$ **então**
    atribuir vetor normal do v_j no conjunto C_d;
  fim
  senão
    $V = v_1, v_2, ..., v_m$;
    atribuir a V vértices com mesma aresta a v_j;
    $i \in 1, .., m$;
    para $\forall v_i \in V$ **faça**
      ObterVérticeVizinhos(v_i,r-1);
    fim
  fim
fim

Definido as vizinhanças, o cálculo referente a variação no vértice central v_j, segue a seguinte definição: vizinhos mais próximos a v_j tem maiores pesos para definir a variação a ser calculada. Segue a fórmula aplicada:

$$k = \frac{\sum_{i=0}^{r-1}((r-i) * (\frac{\sum_{h=0}^{m} \angle(Po,n_h)}{m})}{r} \tag{3.5}$$

3.3 Extração das Linhas de Borda

A extração das linhas no fragmento consiste em percorrer toda a malha de vértices do objeto verificando a variação da normal do mesmo. Para identificação desses trechos foi necessário estimar um valor de limiar de corte 'T' sobre a malha de vértice. Para definir esse liminar, primeiramente tivemos que conhecer o intervalo do valor que o algoritmo calcula para a variação da normal que é definido entre $[0, pi/2]$.

Para compreender os valores da variação é ilustrado na figura 3.3, duas situações; a primeira, como é o valor quando se trata de uma sub-região planar, no outro é mostrado uma situação de borda. Podemos ver que a variação é menor chegando a zero, quando se trate de uma região planar, no outro a variação vai aumentando podendo chegar a $pi/2$.

A escolha para o melhor valor do liminar de corte foi feita de forma empírica, através de teste, fornecendo alguns valores para T, no final dos testes definimos para o $T = 0.4$. Este valor seleciona os melhores possíveis vértices de borda, valores menores disso puluem

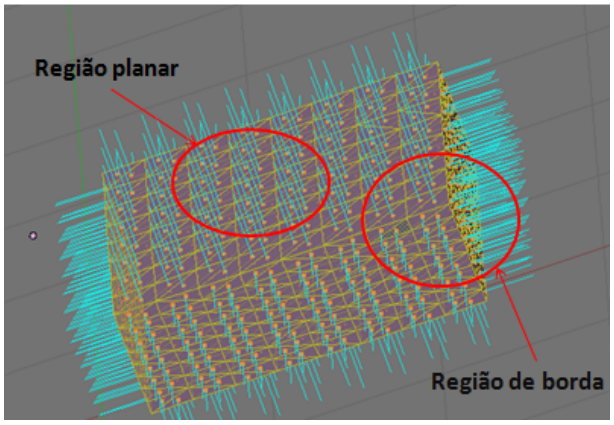

Figura 3.3: Demostração do comportamento das normais na superfície.

com vértices não relevante e valores maiores descartam muito pontos que poderiam ser bordas.

O procedimento de filtrar a malha segue o seguinte raciocínio: dado um conjunto $v = v_1, v_2, v_3...v_n$, para cada item de v verifica-se a variação da normal neste ponto, será selecionado os vértices que possuem valores maiores que T, caso contrário descarta o vértice, no final do processo teremos um conjunto S(vértices selecionados) de pontos. Porém, esse conjunto de pontos não é a linha do fragmento que queremos obter. Para obter esses trechos, tivemos que definir um algoritmo de escolha de vértices, para compor propriamente uma linha.

Dado S, a busca pela linha é definida primeiramente em escolher um ponto de início P_0, esse ponto é o vértice que contém a maior variação de normal, depois de escolher o ponto, verificamos qual o próximo ponto vizinho a P_0, que contém maior variação de normal, este ponto é definido como P_s que será adicionado na linha; essa rotina termina quando não existirem pontos vizinhos. Segue o algoritmo:

Algoritmo 3: ExtrairLinhaBorda

```
Entrada: S , conjuntoLinhasBorda
início
    P_s = seleciona de S vértice com maior variacao de normal;
    repita
        P_s = seleciona de S o vertice vizinho ao P_s com maior variacao;
        adicionarVerticeLinha(conjuntoLinhasBorda,P_s);
        retira o P_s de S;
    até tamanho S != 0;
fim
```

3.4 Comparação de linhas de Fratura 3D

Nesta seção mostraremos como foi tratado o problema de identificação de trechos adjacentes, admitindo que um fragmento contenha, um conjunto de linhas de fraturas, sendo cada uma delas formada por pontos no espaço 3D.

A representação discreta da linha deve conter algumas características importantes para o processo de comparação, dado o segmento $A = a_1, a_2, ..., a_n$ sendo $a_i = (x, y, z)$ e $i \in 0, .., n-1$, o segmento $B = b_1, b_2, ..., b_m$ sendo $b_j = (x, y, z)$ e $j \in 0, .., m-1$, o trecho deve satisfazer os seguintes itens:

- A, B o conjunto não é vazio;
- $A_i \neq A_{i+1}$ e $B_j \neq B_{j+1}$;
- os pontos devem ser consecutivos.

Satisfeitas as condições, podemos manipular os dados das linhas de fratura, a procura de trechos adjacentes de diferentes fragmentos.

Um importante passo para a comparação é que a linha esteja parametrizada, ou seja, os pontos devem estar igualmente espaçados. Para isso, é definido um valor δ para o espaçamento, com esse processo é criado uma super amostra. A topologia da curva original não é mudada, apenas são adicionados mais vértices a amostra da linha para que a mesma seja definida como parametrizada.

O processo de comparação não pode ser feito com pontos reais da curva, pois duas linhas podem ser iguais, porém ter informações diferentes, isso ocorre porque essa linha pode ter sofrido alguma rotação, translação ou ambas. Isso dificulta muito para análise da linha, pois o mesmo pode ser considerado diferente, por causa dos seus dados de posição. Neste caso levamos em conta o uso de invariantes geométricos, então em vez de comparar pontos, vamos comparar a assinatura da linha. Para o projeto consideramos duas invariantes: curvatura e torção; por se tratar de curvas no espaço 3D.

Finalmente, obtemos os dados das linhas que serão usadas para o cálculo, no projeto foi utilizado o algoritmo de força fruta, demonstramos como é feito com o algoritmo abaixo:

Algoritmo 4: CompararLinha

```
Entrada: linhaBordaFrag1 , linhaBordaFrag2
início
    valoresLinha1 = calculaInvariante(linhaBordaFrag1);
    valoresLinha2 = calculaInvariante(linhaBordaFrag2);
    i=1;;
    repita
        trechoAdjacente vazio;
        aux = i;
        repita
            se valoresLinha2(j) = valoresLinha1(aux) então
                adicionaTrechoAdjacente(trechoAdjacente,linhaBordaFrag1(i));
                aux++;
            fim
            senão
                aux = i;
            fim
        até tamanho valoresLinha2 > j;
    até tamanho valoresLinha1 > i;
fim
```

Definimos que para um fragmento ser adjacente a outro, este deve possuir ao menos 10 pontos adjacentes à linha a ser comparada. Em caso de menos pontos, ignoramos, pois pode haver trechos semelhantes não relevantes, que não colaborariam para a solução.

Capítulo 4

Implementação

No trabalho foi implementado na linguagem c++. Neste capítulo apresentaremos, nas seções seguintes, o tipo de arquivo utilizado, a estrutura de dados *halfEdge* e o uso do modelador gráfico blender.

4.1 Tipos de Arquivos

Existem vários tipos de formato de arquivo para armazenar representações tridimensionais, tais como .x3d [17], .ac [18] e .obj[19]. Neste projeto mostraremos o formato *wavefront*.

4.1.1 Formato Wavefront

Nesse trabalho foi usado o formato OBJ(*wavefront object file format*) [19] que é um tipo de arquivo que armazena dados de modelos de objetos 3D. Diversas ferramentas de modelagem, tais como o *3D Studio Max*, *RealWorld Icon Editor* e *Blender* [20] trabalham com essa extensão de arquivo.

Dados de geometria e outras propriedades dos objetos, como as texturas, são armazenados no arquivo OBJ, cujo nome deve ter obrigatoriamente a extensão obj. Informações sobre o material que compõem o objeto são armazenadas em um arquivo MTL correspondente, cujo nome deve ser a extensão mtl. Ambos os arquivos são armazenados como texto ASCII.

Entre os tipos de dados que compõem em um arquivo OBJ, destacam-se:

- os vértices da geometria(v);

- normais aos vértices de textura(vn);
- face(f).

Veja Figura 4.1 abaixo o exemplo de um arquivo que representa um objeto poliédrico.

```
1  # Blender3D v249 OBJ File:
2  # www.blender3d.org
3  mtllib cuboTeste.mtl
4  v 2.391142 0.235835 0.679870
5  v 2.391142 0.235835 2.679870
6  v 0.391142 0.235835 2.679870
7  v 0.391143 0.235835 0.679870
8  v 2.391143 2.235835 0.679870
9  v 2.391142 2.235835 2.679870
10 v 0.391142 2.235835 2.679869
11 v 0.391142 2.235835 0.679870
12 v 2.391142 0.235835 1.679870
13 vn 0.000000 0.000000 -1.000000
14 vn -1.000000 -0.000000 -0.000000
15 vn -0.000000 -0.000000 1.000000
16 vn 1.000000 0.000000 -0.000000
17 vn 1.000000 -0.000000 0.000001
18 vn 0.000000 1.000000 0.000000
19 vn 0.000000 -1.000000 0.000000
20 usemtl Material
21 s off
22 f 46//1 98//1 42//1
23 f 46//1 42//1 8//1
24 f 98//1 74//1 16//1
25 f 98//1 16//1 42//1
26 f 18//1 73//1 98//1
27 f 18//1 98//1 46//1
28 f 73//1 26//1 74//1
29 f 73//1 74//1 98//1
```

Figura 4.1: Exemplo de arquivo no formato .obj

Descrevendo o arquivo temos que as duas primeiras linhas do arquivo contêm comentários sobre a sua geração, e por isto iniciam com # . Na sequência, é descrito o nome do objeto(o), seguindo pelas informações das coordenadas x,y,z de cada vértice(v) do objeto. Depois da listagem dos vértices, começa a listagem dos vetores normais(vn) e das faces(f), nesta ordem. Cada fase é composta por três ou mais vértices. Em geral, uma face composta por três vértices(v1,v2,v3) é representada da seguinte maneira:

f v1/vt/vn v2/vt/vn v3/vt/vn

O conteúdo do arquivo MTL para o objeto representado, contém informações do ambiente tais como: posição da fonte de luz e opacidade.

4.2 Estrutura de Dados

A estrutura da superfície poliédrica é um grande desafio para quem deseja obter informações implícitas sobre ela, pois apenas com informações de vértices, aresta e faces não é possível obter estas informações de forma direta. Por exemplo, dado um objeto queremos efetuar algumas consultas tais como: Achar todas as arestas que incidem em um vértice; achar polígonos que compartilham uma aresta; achar os vizinhos de um polígono.

Para esse fim foram definidas algumas estruturas de dados para armazena essas informações de forma que tais consultas sejam feitas de forma direta, algumas delas são: *Winged-edge*, *Half-Edge* e *Radial-Edge*.

4.2.1 Half-Edge

Half-Edge é uma estrutura de dados que visa otimizar o acesso a estruturas que compõe um objeto 3D composto por vértices, arestas e faces. O conceito básico do seu uso é criar vetores (as *half-edges*) que relacionam dois vértices de uma face. Se uma face possui n vértices, ela deve possui n-1 *half-edges*, igual ao número de arestas. Entretanto, uma *half-edge* é um vetor e possui sentido e direção. Para todas as faces as *half-edges* devem aparentar está indo no mesmo sentido, sendo este horário ou anti-horário.

Além disso, uma *half-edge* deve ter referência a uma outra, chamada de *Twin* (irmãgêmea), que é justamente a associada aos mesmos vértices da primeira, só que pertencente a outra face. Se a *half-edge* seguir a regra de estarem sempre no sentido anti-horário em relação ao vetor normal da face, a sua irmã deverá está no sentido oposto a original. Na figura 4.2 é ilustrado o comportamento da *half-edge* na face.

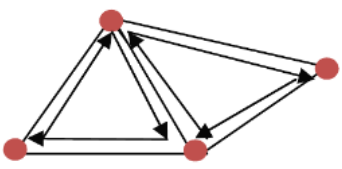

Figura 4.2: Exemplo ilustrativo de faces com half-edge

Assim, uma *half-edge* deve ter referência as seguintes entidades:

- Aos vértices de origem e destino;
- A half-edge irmã;

- A half-edge anterior a ela;
- A half-edge posterior a ela;
- A aresta a qual pertence.

Se a face possui uma referência a pelo menos uma *half-edge*, é possível com isso a partir de uma face encontrar uma *half-edge* e com ela varrer os vértices, as arestas e também as faces adjacentes (já que uma *half-edge* possui uma referência para a irmã, que está na face adjacente).

4.2.1.1 Algoritmo para Construção de uma Half-Edge

Suponha que existe um objeto 3D constituído por vértices agrupados em faces triangulares que possuem uma normal. Os vértices definido como $v_i = (x,y,z)$, onde cada elemento é uma coordenada no eixo 3D e a face as informações da forma (A, B, C, N), onde A, B e C são referências aos vértices que a constituem e N é o vetor normal a face. Com esta configuração, é necessário gerar um conjunto de *Half-Edge* que cubra a malha.

O algoritmo proposto [21] resume-se a varrer todas as faces e gerar as *half-edges* de acordo com a normal da face. Caso uma *half-edge* criada já tenha sua irmã gêmea criada anteriormente, deve-se criar uma referência entre elas.

Assim, tem-se um conjunto (ou lista) de faces ConjV. Devemos varrê-la e constituir as *half-edges* para cada face. Sabe-se já previamente que um vértice é um triângulo constituído por três vértices. Então se sabe também que a face possuirá 3 *half-edges*. Só não sabemos quais vértices constituirão cada *half-edge* nem qual o sentido delas.

Como temos a normal da face, pode-se criar dois vetores que ligam os vértices B ao A ($\overrightarrow{AB}$) e o vértice C ao B ($\overrightarrow{BC}$). Com o produto vetorial deles, cria-se um vetor perpendicular a eles. Se o vetor perpendicular estiver no mesmo sentido da normal da face, quer dizer que os vetores estão com orientação anti-horária, senão estão com orientação horária.

Assim, se o produto vetorial estiver no mesmo sentido da normal, será usado os vetores (mais um terceiro vetor que liga os dois, o $\overrightarrow{CA}$) para construir as *Half-Edges*. Senão, inverte-se os vetores para $\overrightarrow{BA}$ e $\overrightarrow{CB}$ (mais um terceiro, $\overrightarrow{AC}$) e os usa na construção das *half-edge*.

Além disso, é preciso criar uma ligação entre as *half-edges*, qual a sua anterior e

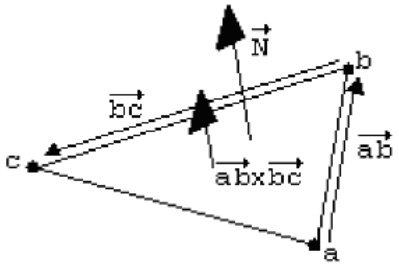

Figura 4.3: Detectar a orientação das possíveis half-edges

sua posterior. Por exemplo, as *half-edges*, $\overrightarrow{AB}$, $\overrightarrow{BC}$ e $\overrightarrow{CA}$ devem possuir a seguinte configuração:

Tabela 4.1: tabela Half-Edge

Half-Edge	**Anterior**	**Posterior**
$\overrightarrow{AB}$	$\overrightarrow{CA}$	$\overrightarrow{BC}$
$\overrightarrow{BC}$	$\overrightarrow{AB}$	$\overrightarrow{CA}$
$\overrightarrow{CA}$	$\overrightarrow{BC}$	$\overrightarrow{AB}$

Por último, é preciso saber se existe uma half-edge irmã a half-edge recém-criada. Isto é, se uma half-edge $\overrightarrow{AB}$ foi criada, é preciso saber se sua inversa $\overrightarrow{BA}$ já foi criada, se sim, é preciso fazer a ligação entre elas, considere $\overrightarrow{x}$ como *half-edges* irmã a outra. Agora já podemos propor um algoritmo para a criação das half-edges:

Algoritmo 5: Criar Half-Edge

Entrada: Conjuto de faces com 3 vértices.
Saída: Half-Edge
início
  para $F \subset ConjF$ **faça**
    $Criarvetores$:
    $\overrightarrow{Ha} = \overrightarrow{AB}$;
    $\overrightarrow{Hb} = \overrightarrow{CB}$;
    $\overrightarrow{Hc} = \overrightarrow{CA}$;
    $\overrightarrow{ProdutoInterno} = \overrightarrow{Ha} \text{x} \overrightarrow{Hb}$;
    se *não for o mesmo sentido*$(\overrightarrow{ProdutoInterno}, \overrightarrow{N})$ **então**
      $\overrightarrow{Ha} = \overrightarrow{BA}$;
      $\overrightarrow{Hb} = \overrightarrow{CB}$;
      $\overrightarrow{Hc} = \overrightarrow{AC}$;
    fim
    se *já foi criada irmã* $\vec{x}$ *de* $\overrightarrow{Ha}$ **então**
      atribui $\vec{x}$ como irmã de $\overrightarrow{Ha}$ e $\overrightarrow{Ha}$ como irmã de $\vec{x}$;
    fim
    se *já foi criada irmã* $\vec{x}$ *de* $\overrightarrow{Hb}$ **então**
      atribui $\vec{x}$ como irmã de $\overrightarrow{Hb}$ e $\overrightarrow{Hb}$ como irmã de $\vec{x}$;
    fim
    se *já foi criada irmã* $\vec{x}$ *de* $\overrightarrow{Hc}$ **então**
      atribui $\vec{x}$ como irmã de $\overrightarrow{Hc}$ e $\overrightarrow{Hc}$ como irmã de $\vec{x}$;
    fim
    atribui $\overrightarrow{Hb}$ half-edge posterior de $\overrightarrow{Ha}$;
    atribui $\overrightarrow{Hc}$ half-edge posterior de $\overrightarrow{Hb}$;
    atribui $\overrightarrow{Ha}$ half-edge posterior de $\overrightarrow{Hc}$;
    atribui $\overrightarrow{Hc}$ half-edge posterior de $\overrightarrow{Ha}$;
    atribui $\overrightarrow{Ha}$ half-edge posterior de $\overrightarrow{Hb}$;
    atribui $\overrightarrow{Hb}$ half-edge posterior de $\overrightarrow{Hc}$;
    adiciona $\overrightarrow{Ha}$, $\overrightarrow{Hb}$ e $\overrightarrow{Hc}$ ao container de saída das half-edges.
  fim
fim

Capítulo 5

Testes

5.1 Objetos Virtuais

Nos testes apresentados aqui mostraremos os objetos modelados no blender. Para efetuação de um teste é necessário que o objeto esteja representado no modelo *wavefront* do formato .obj.

O modelador gráfico [20] blender3d é um programa de computador de código aberto, desenvolvido pela Blender Foundation, para modelagem, animação, texturização, composição, renderização, edição de vídeo e criação de aplicações interativas em 3D, tais como jogos, apresentações e outros,o programa é multiplataforma, estando portanto disponível para diversos sistemas operacionais.

5.1.1 Dados de Entrada

A entrada de dados é composto por 2 fragmentos tridimensionais ilustradas nas figuras 5.1 e 5.2 abaixo.

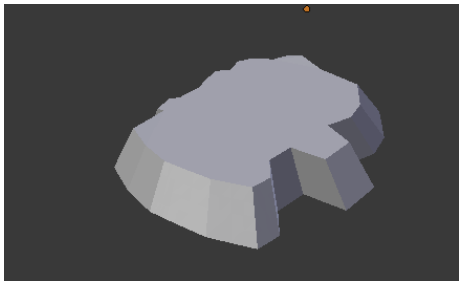

Figura 5.1: Fragmentos 1

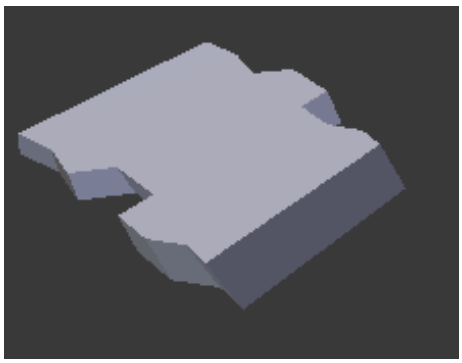

Figura 5.2: Fragmento 2

Dados Fragmento		
Fragmento	Vértices	Faces
1	5.561	11.321
2	4.583	10.707

Tabela 5.1: Demostração dos dados modelados.

5.1.2 Obtenção das bordas dos fragmentos

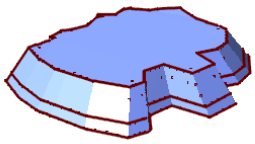

Figura 5.3: Bordas Fragmento 1

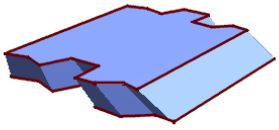

Figura 5.4: Bordas Fragmento 2

Neste teste vimos a identificação e extração das linhas de bordas dos objetos virtuais, que serão importantes para o próximo passo, a identificação de trechos adjacentes.

5.1.3 Comparação de fragmentos

Neste projeto, só foram feitos testes de comparação em objetos virtuais, segue abaixo o exemplo ilustrativo da saída do programa:

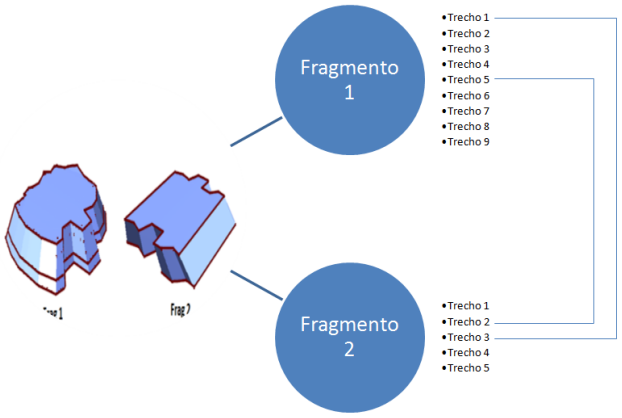

Figura 5.5: Resultados da comparação de trechos.

5.2 Objetos Reais

Para os testes de identificação de borda, utilizamos uma moringa de cerâmica que foi quebrado em vários fragmentos. O primeiro passo consiste em obter os dados tridimensionais dos mesmos, para este fim, foi usado o programa estereofotométrico para obter os dados tridimensionais através de imagens do fragmento. A seguir, serão mostrados os dados de entrada para o programa, e os arquivos de saída que contém informações tridimensionais da imagem.

5.2.1 Dados de Entrada

Tiramos algumas fotos do fragmento, sobre a mesma perspectiva, alterando apenas a localização da luz, segue a figura 5.6. É necessário também usar o gabarito (objeto esférico lambertiano, cujas normais são conhecidas), junto ao objeto que queremos reconstruir.

Após o processo sobre as imagens, o programa gera como dado de saída, dois arquivos, um contendo os dados de z para cada pixel da imagem, segue o exemplo do arquivo na figura 5.7, e outro arquivo denominado arquivo de peso, que define quais dados de Z são significantes, informações de fundo e do gabarito são considerados irrelevantes, como ilustrado na figura 5.8.

Figura 5.6: Imagem dos fragmentos, com três gabaritos.

```
begin float_image_t (format of 2006-03-25)
NC = 1
NX = 756
NY = 531
     0      0 -1.2004021e+02
     1      0 -1.2003143e+02
     2      0 -1.1996709e+02
     3      0 -1.0597701e+02
     4      0 -9.5277588e+01
     5      0 -8.6553673e+01
     6      0 -7.9181534e+01
     7      0 -7.2817390e+01
     8      0 -6.7247810e+01
     9      0 -6.2327938e+01
    10      0 -5.7952873e+01
    11      0 -5.4042881e+01
    12      0 -5.0535172e+01
    13      0 -4.7378929e+01
    14      0 -4.4532162e+01
    15      0 -4.1959587e+01
    16      0 -3.9631187e+01
    17      0 -3.7521126e+01
    18      0 -3.5606987e+01
    19      0 -3.3869175e+01
    20      0 -3.2290447e+01
    21      0 -3.0855549e+01
    22      0 -2.9550928e+01
    23      0 -2.8364494e+01
    24      0 -2.7285412e+01
    25      0 -2.6303949e+01
    26      0 -2.5411335e+01
    27      0 -2.4599642e+01
    28      0 -2.3861694e+01
    29      0 -2.3190979e+01
    30      0 -2.2581575e+01
```

Figura 5.7: Arquivo de altura, para cada ponto é definido o z.

Figura 5.8: Arquivo de peso, os pontos preto são considerados como irrelevantes.

5.2.2 Transformação dos dados para formato .obj

Para trabalhar com os dados de saída, fornecidos pelo programa esteofotométrico, tivemos que criar um tradutor para transformar esses dados em formato de arquivo .obj(*wavefront*), a seguir vemos o resultado.

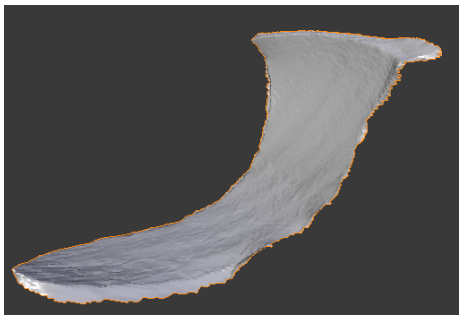

Figura 5.9: Frag 1

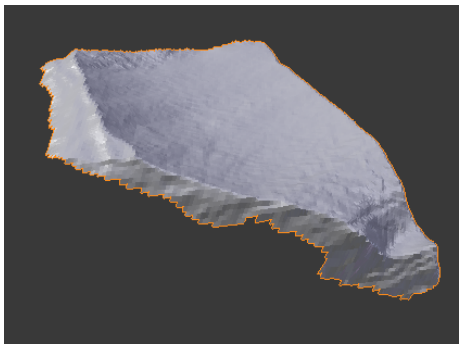

Figura 5.10: Frag 2

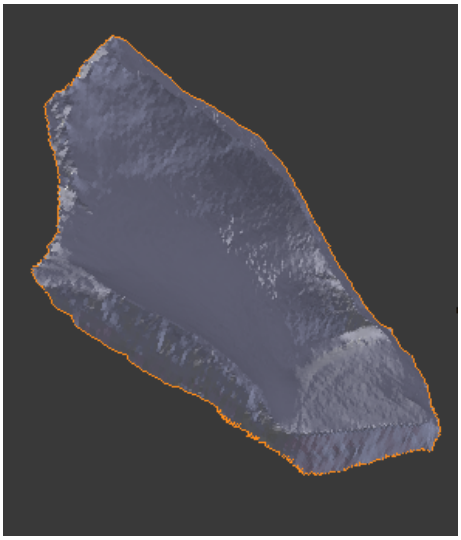

Figura 5.11: Frag 3

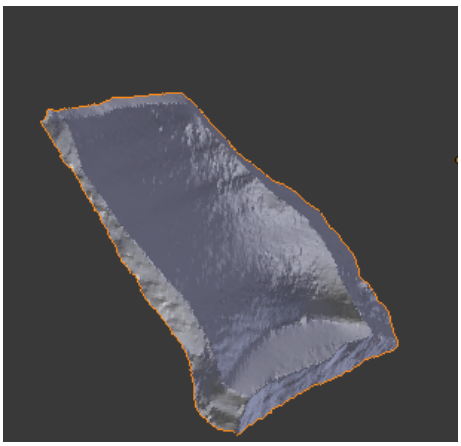

Figura 5.12: Frag 4

Dados Fragmento		
Fragmento	Vértices	Faces
1	14.883	29.010
2	11.274	21.990
3	22.106	43.550
4	15.487	35.517

Tabela 5.2: Apresentação dos fragmentos na estrutura de vértices e faces.(*wavefront*)

5.2.3 Obtenção das bordas dos fragmentos

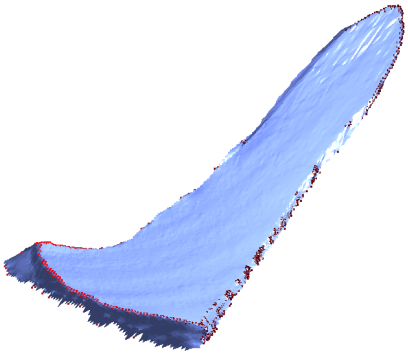

Figura 5.13: Bordas do fragmento 1

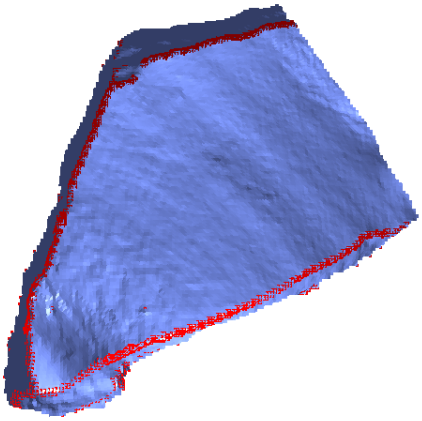

Figura 5.14: Bordas do fragmento 2

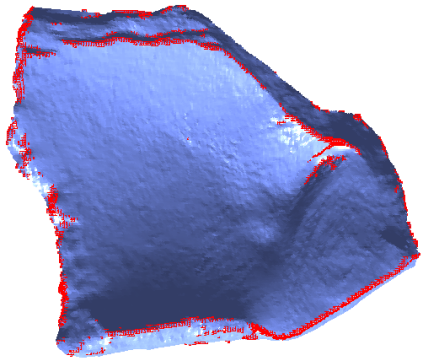

Figura 5.15: Bordas do fragmento 3

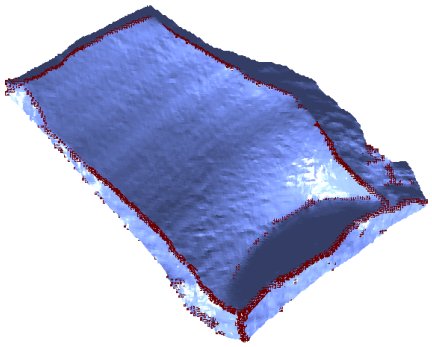

Figura 5.16: Bordas do fragmento 4

Neste teste foi demonstrada a identificação e extração das linhas de bordas das representações dos fragmentos reais.

Capítulo 6

Conclusões

6.1 Conclusão

Neste projeto apresentamos métodos para identificação de bordas de fragmentos, utilizando as informações da normal dos vértices para os cálculos, e verificamos que os vértices que possuíam as maiores variações de normais eram os vértices que faziam parte da borda dos elementos. Utilizamos invariantes geométricos dos objetos, tais como curvatura e torção, que foram utilizados apenas em objetos virtuais para comparação de trechos adjacentes, o uso em objetos reais exige maior estudo sobre melhor obtenção da linha de fratura.

Mostramos o uso e implementação da estrutura de dados *HalfEdge*, muito importantes para processar os dados dos fragmentos em 3D. O uso dessa estrutura possibilitou o acesso rápido aos dados na malha de vértices e arestas. Outra contribuição foi a criação do tradutor dos dados estereofotométrico e mapa de alturas para o formato *wavefront*.

Existem alguns problemas ainda a serem resolvidos como a necessidade de melhorar a escolha dos trechos de borda, porém para a maioria dos casos de testes em objetos virtuais a técnica utilizada apresentou resultados satisfatórios.

6.2 Trabalhos Futuros

Deve ser feito um aprimoramento sobre o processo da escolha de trechos, usando técnicas de regressão linear, usando *best curve fitting*, isso ajudaria a filtrar as linhas de borda dos fragmento para retirar possíveis ruídos.

O uso de programação dinâmica para os cálculos de comparação de trechos reduziria

o custo em relação ao método de força bruta, atualmente usado no neste trabalho. O projeto identifica trechos locais adjacentes, o próximo passo seria a reconstrução total do objeto, fazendo comparações globais.

APÊNDICE A – Implementação Half-Edge

A implementação realizada possui três classes principais: A classe Half-edge, que implementa a estrutura Half-edge. A classe HalfEdgeGraph armazena as Half-Edges e HalfEdgeContainer implementa o algoritmo de construção da representação half-edge a partir de um modelo 3D.

Nosso modelo 3D é constituído pelas estruturas VERT (vértices ou vetores), OBJ (objeto, composto por VERT e FACE) e FACE (Face composta por VERT).

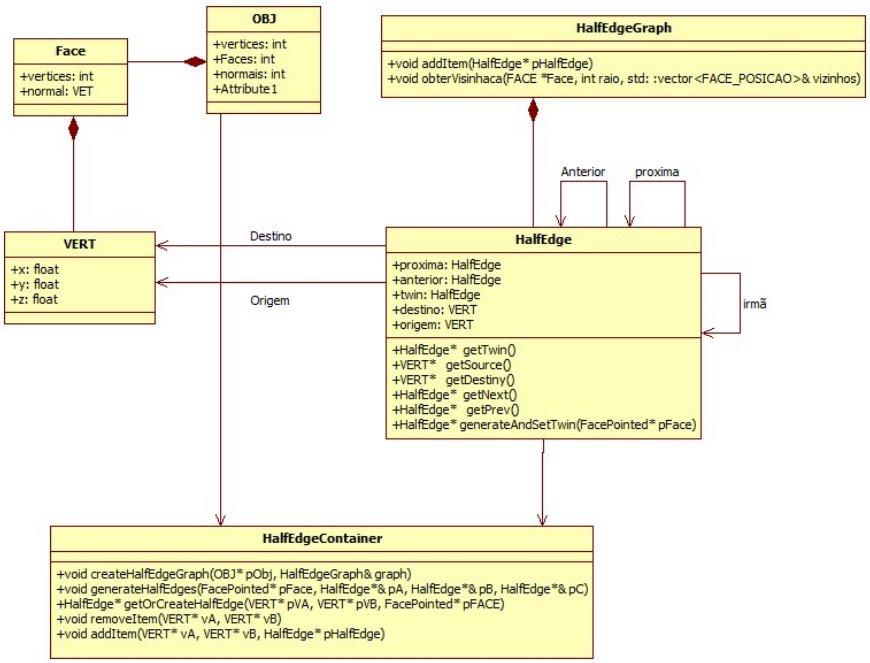

Figura A.1: Diagrama de classe da Half-Edge

Para realizar a tarefa, o sistema deve carregar um modelo poliédrico e criar um objeto de HalfEdgeContainer e outro de HalfEdgeGraph. Depois se deve passar para o método geraHalfEdge o poliédrico e o HalfEdgeGraph.

Referências

[1] SARACCHINI, R. F. V.; MORAIS, J. F. Um método estereofotométrico para reconstrução tridimensional de objetos. 2006.

[2] LEITãO, H. C. G.; STOLFI, J. *Pesquisa em Arqueologia Computacional*. 2008. [Online; accessed 30-Junho-201-0]. Disponível em: <http://www.ic.uff.br/ raab>.

[3] LEITAO, H. C. da G.; STOLFI, J. A multiscale method for the reassembly of two-dimensional fragmented objects. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, v. 24, n. 9, p. 1239 – 1251, sep 2002. ISSN 0162-8828.

[4] ÜçOLUK, G.; TOROSLU, I. H. Automatic reconstruction of broken 3-d surface objects. *Computers and Graphics*, v. 23, n. 4, p. 573 – 582, 1999. ISSN 0097-8493. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0097849399000758>.

[5] KAMPEL, M.; SABLATNIG, R. On 3d mosaicing of rotationally symmetric ceramic fragments. *Pattern Recognition, International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, p. 265–268, 2004. ISSN 1051-4651.

[6] KONG, W.; KIMIA, B. B. On solving 2d and 3d puzzles using curve matching. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 2, p. 583, 2001. ISSN 1063-6919.

[7] HUANG, Q.-X. et al. Reassembling fractured objects by geometric matching. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 25, p. 569–578, July 2006. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/1141911.1141925>.

[8] WILLIS, A.; COOPER, D. Bayesian assembly of 3d axially symmetric shapes from fragments. *Computers Graphics*, p. 82 – 89, 2004. ISSN 1063-6919. Disponível em: <http://ieeexplore.ieee.org/xpl/freeabsall.jsp.arnumber=1315017>.

[9] CARVALHO, P. C. P. *Introdução à Geometria Espacial*. [S.l.]: IMPA-VITAE, 1993.

[10] LIMA, E. L. *Coordenadas no Espaço*. [S.l.]: IMPA-VITAE, 1992.

[11] WIKIPéDIA. *Desigualdade de Cauchy-Schwarz*. 2010. [Online; accessed 21-Junho-2010]. Disponível em: <http://www.pt.wikipedia.org/wiki/Desigualdade_de_Cauchy-Schwarz>.

[12] WIKIPéDIA. *Desigualdade triangular*. 2010. [Online; accessed 21-Junho-2010]. Disponível em: <http://www.pt.wikipedia.org/wiki/Desigualdade_triangular>.

[13] MUNDY, J. L.; ZISSERMAN, A. (Ed.). *Geometric invariance in computer vision*. Cambridge, MA, USA: MIT Press, 1992. ISBN 0-262-13285-0.

[14] LEWIER, T. et al. Curvature and torsion estimators based on parametric curve fitting. *Elsevier*, 2005.

[15] DOERMANN, D. S.; WEISS, E. R. I. Logo recognition using geometric invariantes. *Citeseer*, 1993.

[16] UM algoritmo para identificação de linhas de borda baseado na variação do mapa de normais. *Anais do CNMAC*, 2009.

[17] THE3DSTUDIO. *3D Models, Stock Photos & Images, Texture*. 1996. [Online; accessed 11-Maio-2010]. Disponível em: <http://www.the3dstudio.com>.

[18] INIVIS. *3D Disign Software*. 2010. [Online; accessed 11-Maio-2010]. Disponível em: <http://www.inivis.com>.

[19] MANSSOUR, M. C. I. H. *OpenGL uma abordagem prática e objetiva*. [S.l.]: Novatec, 2006.

[20] BLENDER. *Blender*. 2010. [Online; accessed 25-Julho-2011]. Disponível em: <http://www.blender.org>.

[21] COMPUTATIONAL Geometry:Algorithms and Applications. *Third Edition.Springer*, 2008.