UNIVERSIDADE FEDERAL DO PARANÁ
Bacharelado em Ciência da Computação

Qualidade e Comparação de Malhas Triangulares

Trabalho referente ao Trabalho de Graduação II do curso de Bacharelado em Ciência da Computação da Universidade Federal do Paraná. Orientado pelo professor André Luiz Pires Guedes.

Realizado pelos alunos:

Flávio Augusto Coraiola
Juliana Izabel Mendes
Vitor Born Abreu

CURITIBA
OUTUBRO DE 2008.

Sumário

1. Introdução

Nesse trabalho elaboramos um programa que faz medições e comparações entre malhas de triângulos. Para tal, faremos duas avaliações baseadas nas medidas dos triângulos e duas comparações que irão ajudar a decidir qual malha é melhor para ser utilizada.

As malhas de triângulos são estudadas por suas aplicações na computação gráfica. São uma das representações de dados espaciais mais utilizadas, pois possibilitam a manipulação e visualização de superfícies de alta complexidade, além de apresentarem diversas vantagens, como suporte direto em software e hardware e maior velocidade e simplicidade. A transformação de conjuntos de dados espaciais distintos, entre eles modelos de terrenos, conjuntos de pontos tridimensionais e dados volumétricos, em malhas triangulares é amplamente estudado [1].

A motivação encontrada para o desenvolvimento do trabalho é a necessidade de comparar as malhas de um mesmo objeto, verificando qual delas é a melhor para determinada aplicação. Existem vários programas que fazem triangulações, usando diferentes métodos de construção de malhas, que podem gerar modelos apresentando diferentes conjuntos de pontos e triângulos. Esse estudo ajuda a definir se sua saída é de boa qualidade ou não para o que queremos utilizar. O programa compara duas malhas criadas por diferentes fontes e, possivelmente, com diferente número de triângulos. Dependendo do objetivo, pode ser melhor usar uma malha com mais ou menos detalhes, por isso é importante saber se o que elas representam é o mesmo objeto ou algo muito próximo disso.

No capítulo 2, definiremos alguns conceitos utilizados ao longo do trabalho. No capítulo 3, mostramos como funcionam os algoritmos de medição dos triângulos das malhas e no capítulo 4, os algoritmos de comparação entre malhas. No capítulo 5 exibimos os testes realizados com as malhas triangulares.

2. Conceitos

Malhas poligonais - Malhas poligonais são amplamente utilizadas para modelos geométricos, devido ao fato de serem flexíveis e suportadas pela maioria dos pacotes de modelagem e visualização. Uma malha poligonal é uma superfície ou objeto tridimensional representado por meio de um conjunto de polígonos [2]. Cada polígono é constituído de vértices (pontos com localização tridimensional), arestas (que determinam quais vértices estão conectados), e faces (delimitadas pelas arestas). O agrupamento dos diversos polígonos totaliza a representação do objeto ou superfície em questão.

O tipo mais utilizado de malha poligonal é formada por triângulos. Malhas triangulares, também chamadas de triangulações, são mais rápidas, mais simples de representar e mais fáceis de trabalhar que outros tipos de modelagem, além de serem suportadas por diversos softwares de modelagem e visualização e possuir até mesmo suporte direto em hardware.

Malhas triangulares são uma representação compacta, pois os triângulos comprimem eficientemente dados redundantes, tipicamente pontos pertencentes ao mesmo plano. O triângulo é o polígono coplanar mais simples (todos os pontos do triângulo estão no mesmo plano). Isso simplifica a matemática necessária para executar todos os cálculos. Exemplos de malhas de triângulos são encontrados nas figuras 2.1 e 2.2.

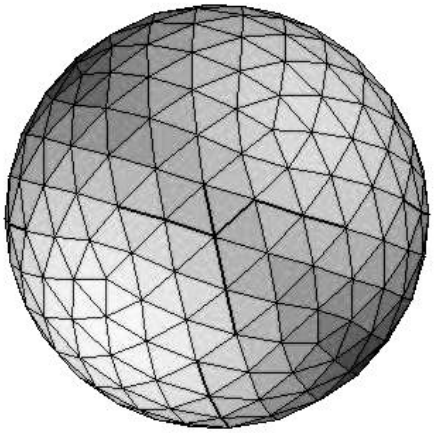


Fig. 2.1 - Esfera

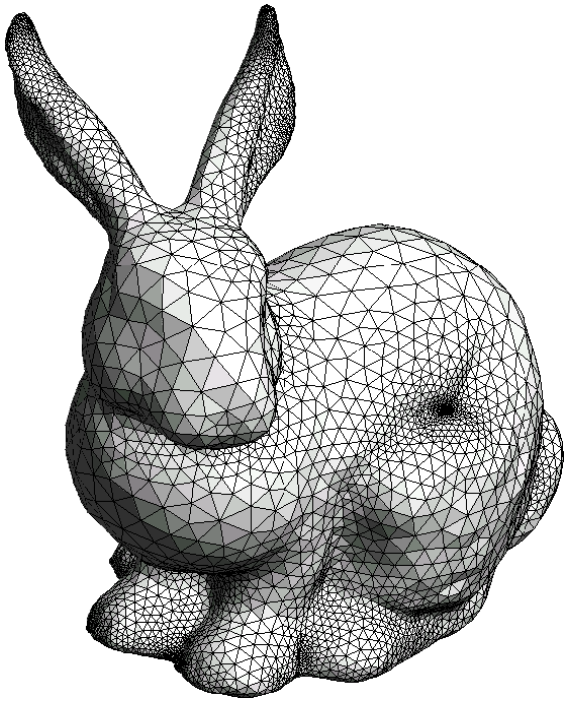


Fig. 2.2 - Stanford Bunny

Triangulação – É uma decomposição de uma superfície em triângulos, e tem as seguintes propriedades:
(1) Qualquer aresta é aresta de exatamente dois triângulos (Fig. 2.3);

(2) Qualquer vértice, v, é o vértice de pelo menos três triângulos, e todos os triângulos tendo v como vértice se dispõe em um ciclo em seu redor (Fig. 2.4).

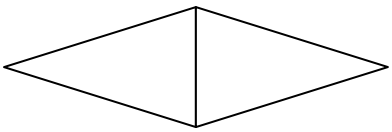


Fig. 2.3

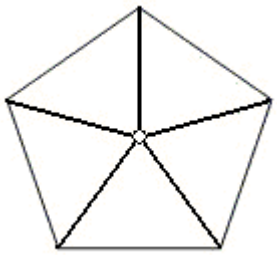


Fig. 2.4

A grande vantagem de uma triangulação é que ela reduz nosso objetivo de classificação a um problema combinatório finito, que podemos então atacar com Matemática Finita [3].

Qualidade dos Triângulos – A qualidade dos triângulos é analisada medindo o menor ângulo (entre 0 e 60 graus) de cada triângulo que compõe a malha. Esta medida permite obter uma idéia geral sobre a forma de cada triângulo: se o menor ângulo for próximo de 60 graus então o triângulo está próximo de ser eqüilátero. Os triângulos com essa característica são melhores porque são menos propensos a erros numéricos, que podem ser causados pela dificuldade em medir um triângulo com altura muito pequena (representados na figura 2.5).

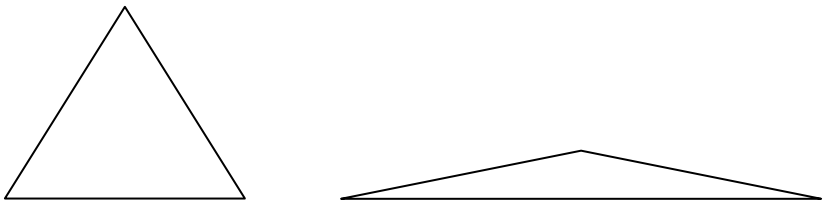


Fig. 2.5

A triangulação de Delaunay pode ser utilizada, pois possui a propriedade de construir os maiores triângulos possíveis, todos aproximadamente eqüiláteros (O´Rourke, 1993), o que é desejável, pois assim são evitados problemas de interpolação e artefatos visuais [4].

Medidas típicas da qualidade da triangulação analisam o maior ou o menor dos ângulos, a razão entre a menor e a maior de suas arestas, a razão entre os raios dos círculos inscrito e circunscrito etc., tendo por parâmetro a relação equivalente no triângulo eqüilátero. A medida de qualidade da malha é dada pelo elemento de pior medida [5].

Distância entre dois pontos – A distância euclidiana é um conceito matemático que representa a menor distância existente entre dois pontos na Geometria Euclidiana. Quanto menor o valor da distância euclidiana entre dois pontos, mais próximos eles se apresentam no espaço. A distância euclidiana entre X e Y é dada por:

$d(P_1,P_2) = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$,

e é mostrada na figura 2.6.

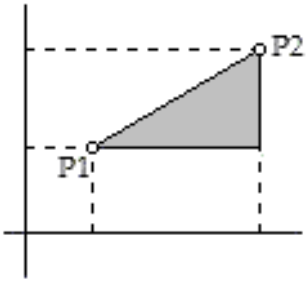


Fig. 2.6

Distância de ponto a plano - Seja P um ponto localizado fora de um plano. A distância do ponto ao plano é a medida do segmento de reta perpendicular ao plano em que uma extremidade é o ponto P e a outra extremidade é o ponto que é a interseção entre o plano e o segmento. Se o ponto P estiver no plano, a distância é nula [6]. A figura 2.7 mostra essa distância.

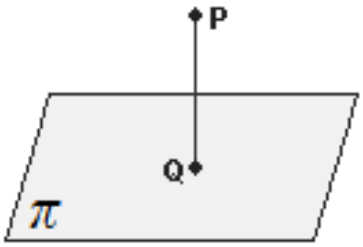


Fig. 2.7

Para calcular a distância, usamos a fórmula do vetor projeção ortogonal:

$$\text{dist}(P,\pi) = \left\| \text{proj}_N \overrightarrow{PQ} \right\| = \frac{\left| \overrightarrow{PQ} \cdot N \right|}{\left\| N \right\|}$$

Onde: PQ é um vetor entre um ponto Q qualquer do plano π, e o ponto P, e N é o vetor normal do plano. PQ é projetado ortogonalmente no vetor normal ao plano e essa será a distância do ponto ao plano [7].

Histograma – Representa os valores obtidos através das medições de triângulos e das comparações entre malhas de triângulos.

3. Descrição do algoritmo de qualidade das malhas

3.1 Avaliações

As avaliações são baseadas na razão entre as medidas do círculo inscrito e circunscrito de cada triângulo e na medida dos ângulos internos dos triângulos.

Na primeira, o algoritmo calcula o tamanho dos lados do triângulo a partir dos seus pontos, e com isso, calcula o raio do círculo circunscrito e do círculo inscrito ao triângulo.

O raio do círculo circunscrito é dado pela fórmula:

$$R = \frac{a}{2 \operatorname{sen} \alpha}$$

Onde a é um dos lados do triângulo e α é o ângulo oposto a esse lado.

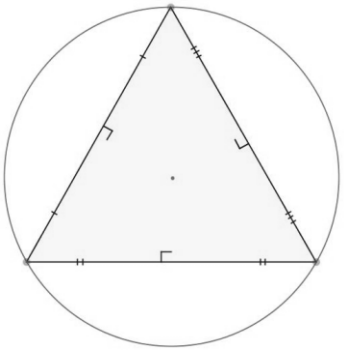
O raio do círculo inscrito é dado pela fórmula:

$$r = 4R \operatorname{sen} \frac{\alpha}{2} . \operatorname{sen} \frac{\beta}{2} . \operatorname{sen} \frac{\gamma}{2}$$

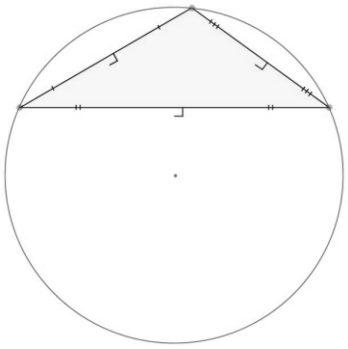
Onde a, b e c são os lados do triângulo e α, β e γ são os ângulos opostos à a, b e c, respectivamente.

Em seguida, o programa calcula a razão entre os raios do círculo inscrito e circunscrito. Quanto maior é essa razão, pior é a qualidade do triângulo em questão.

Vemos a seguir a diferença entre os círculos circunscritos de um triângulo eqüilátero e de um escaleno com a altura relativamente pequena:



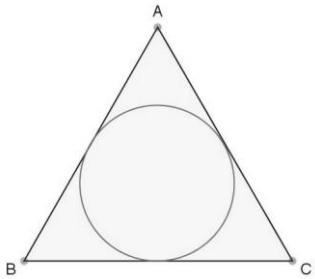
Triângulo eqüilátero
circunscrito



Triângulo escaleno
circunscrito

Fig. 3.2

Vemos a seguir a diferença entre os círculos inscritos de um triângulo eqüilátero e de um escaleno com a altura relativamente pequena:

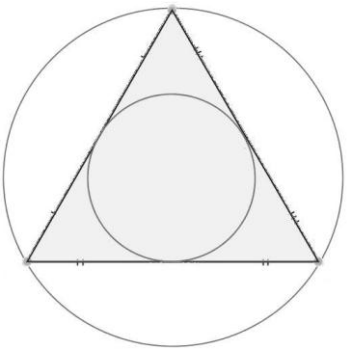


Triângulo eqüilátero
inscrito

Triângulo escaleno
inscrito

Fig 3.3

A comparação entre os círculos pode ser vista na figura 3.4.

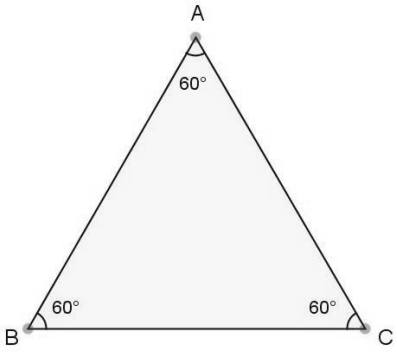


Círculos inscrito e circunscrito
em um triângulo eqüilátero

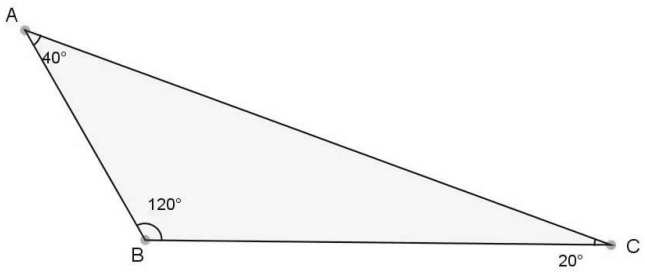
Fig. 3.4

A segunda avaliação usa a medida do menor ângulo interno de um triângulo. Para isso, o algoritmo calcula o tamanho dos lados do triângulo a partir dos seus pontos, e com isso, obtém o valor do cosseno de cada ângulo. Com o maior cosseno, retorna o tamanho do menor ângulo do triângulo em questão. Quanto menor for o tamanho do menor ângulo interno do triângulo, pior será sua qualidade.

Vemos na figura 3.5 a diferença da medida dos ângulos internos de um triângulo eqüilátero e de um obtusângulo.



Ângulos internos do triângulo eqüilátero



Ângulos internos do triângulo obtusângulo

Fig. 3.5

4. Descrição do algoritmo de comparações de malhas

4.1 Comparações

As comparações são feitas com base no cálculo da distância total, distância relativa, menor distância e maior distância entre duas malhas de triângulos, baseado na menor distância entre os pontos de duas malhas e entre os pontos e os planos formados pelos triângulos das malhas.

A primeira comparação é feita com a distância ponto a ponto, ela é mais rápida e melhor para malhas que se supõe que tenham uma definição parecida, isto é, com uma quantidade de triângulos semelhante, porém arranjados de forma diferente. A desvantagem é que, caso uma malha seja mais detalhada que a outra, com a quantidade de triângulos muito maior, a distância ponto a ponto pode ser uma distância entre um triângulo contido em outro, ou um ponto de um triângulo dentro de outro triângulo, e o algoritmo calcularia uma distância diferente de zero.

A segunda comparação é feita com a distância de ponto a plano, sua vantagem é que determina a diferença entre duas malhas com arranjos de triângulos diferentes, ou com uma definição muito diferente, por exemplo, um coelho feito com 1000 triângulos, e um coelho semelhante feito com um bilhão de triângulos, devem ter uma diferença de arranjo pequena, apesar de o segundo ter muito mais detalhes. Essa comparação considera que um ponto contido em um plano é pertencente a algum triângulo, e que um ponto pouco distante do triângulo, mas próximo ao centro, tem distância diferente de zero, porém pequena. Sua desvantagem é o custo mais alto com relação a velocidade.

A comparação de malhas é importante para tomar decisões sobre seu uso. Por exemplo, um mesmo objeto pode ser representado por várias malhas diferentes. O programa ajuda a

decidir qual delas tem os triângulos de melhor qualidade e se elas estão realmente representando o mesmo objeto, mostrando quão diferentes elas são entre si. Nesse caso, as malhas podem ter diferentes resoluções, e dependendo do objetivo da malha, pode ser desnecessário usar uma malha com mais triângulos (que seria mais pesada e demorada para carregar, utilizando mais recursos computacionais) se uma mais simples já seria o suficiente para o que se quer desenvolver.

As figuras 4.1 e 4.2 mostram três malhas semelhantes com diferente número de triângulos e como elas podem ser utilizadas (exemplo retirado de [1]).

A avaliação dos triângulos e a comparação das malhas terão seus resultados exibidos num histograma, como mostra o exemplo a seguir:

Grafico da qualidade dos triangulos, de acordo com o menor angulo.

Quantidade:	2	102	406	653	794	876	835	696	490	146
Nota:	1	2	3	4	5	6	7	8	9	10

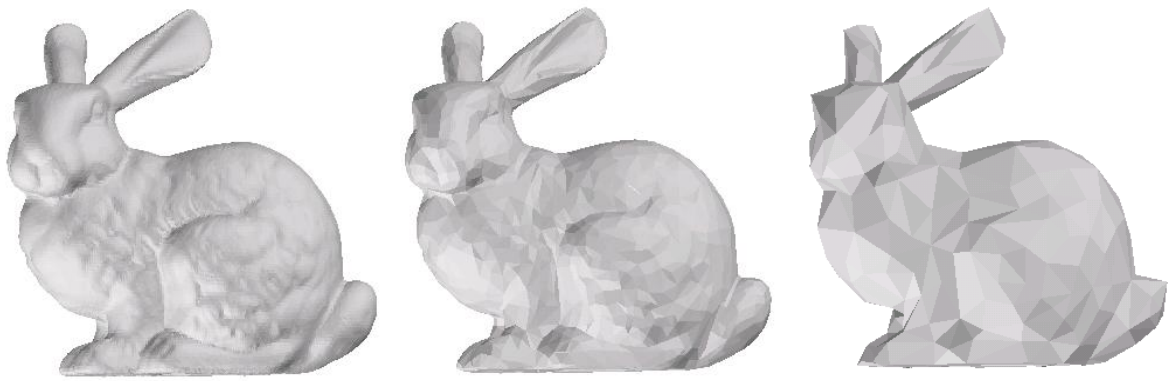


Fig. 4.1

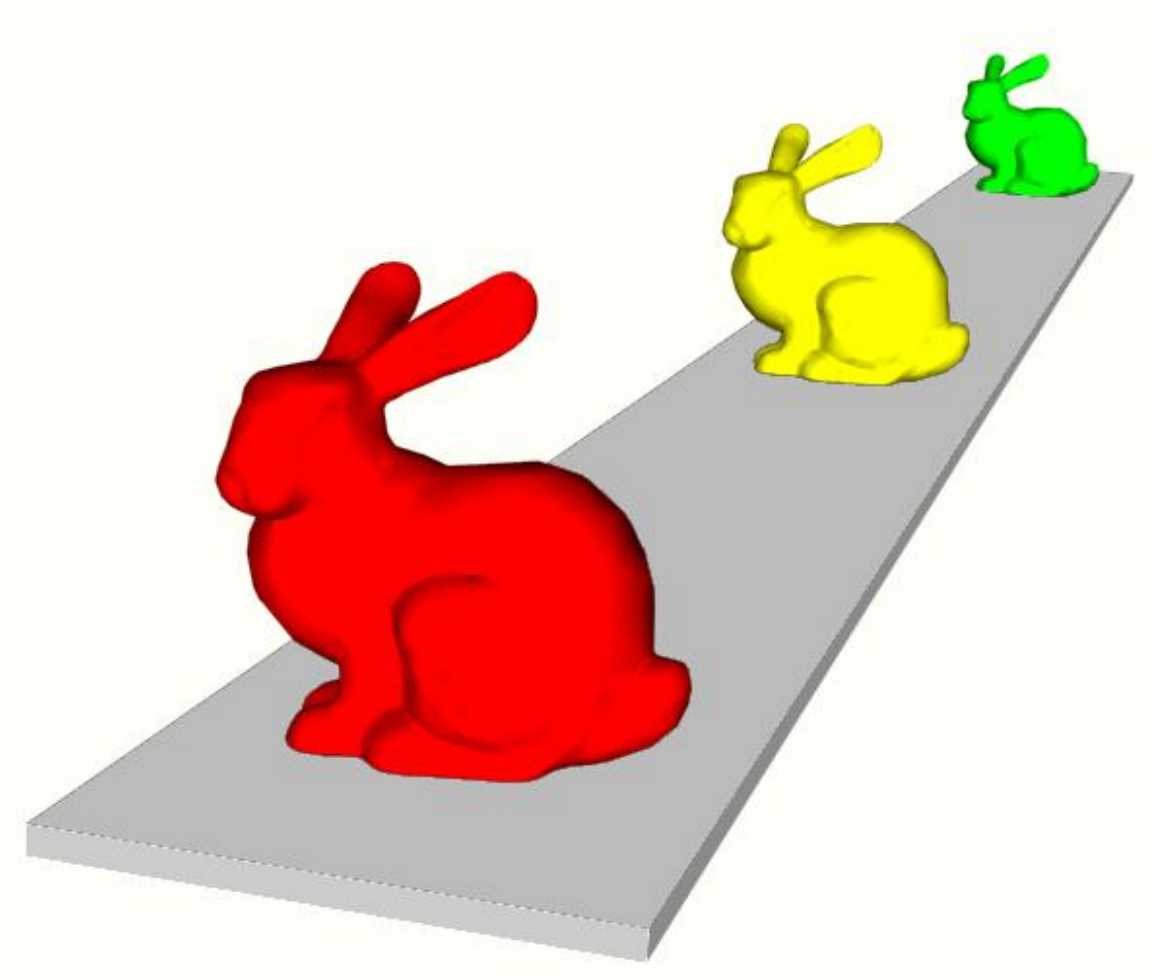


Fig. 4.2

4.2 Descrição do funcionamento

O algoritmo varre o arquivo da malha, armazenando os valores dos vértices e dos triângulos. Calcula as razões entre o círculo inscrito e o círculo circunscrito dos triângulos, calcula os menores ângulos de cada triângulo, calcula a distância total, distância relativa, menor distância e maior distância entre duas malhas de triângulos de duas formas: comparação de todos os pontos de uma malha com todos da outra e comparação dos planos de uma malha com os pontos da outra.

O programa foi escrito em linguagem C. O formato de arquivo aceito pelo algoritmo é OBJ. A descrição da implementação pode ser encontrada no Anexo I, ao final do documento.

5. Testes

Neste capítulo são apresentados o formato de arquivo utilizado e os testes realizados utilizando malhas de triângulos para mostrar o comportamento do programa ao longo da sua execução.

5.1 Formato OBJ

O formato utilizado para as malhas é uma simplificação do OBJ, que foi desenvolvido pela Wavefront para especificar facilmente objetos representados por polígonos. A idéia principal do formato é apresentar inicialmente uma lista de vértices seguida de uma lista de faces que apontam para os vértices que a compõe [8].

Os arquivos OBJ não exigem qualquer tipo de cabeçalho, embora seja comum iniciar o arquivo com uma linha de comentário. Linhas de comentário iniciam com #. Linhas e espaços em branco podem ser livremente acrescentados no arquivo para ajudar na formatação e legibilidade. Um exemplo é exibido na figura 3.1.

O formato de arquivo OBJ suporta linhas, polígonos, curvas e superfícies. Linhas e polígonos são descritos em função dos seus pontos, enquanto curvas e superfícies são definidas com pontos de controle e outras informações, dependendo do tipo de curva [9].

Os tipos de dados suportados por esse formato e suas palavras-chave (entre parênteses) são:

Vértices: vértices geométricos (v), vértices texturizados (vt), vértices normais (vn), vértices de parâmetro no espaço (vp);

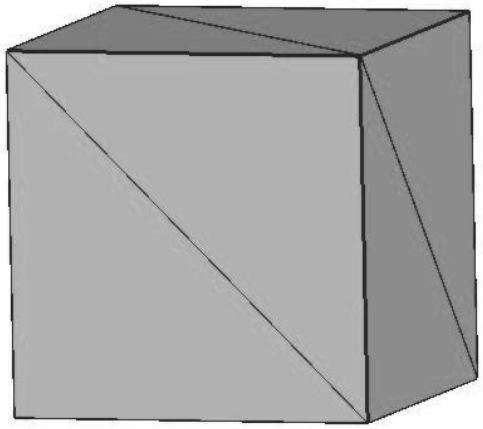
Atributos de forma-livre (curvas e superfícies): graus (deg), matriz básica (bmat), tamanho da etapa (step), tipo curva ou superfície (cstype);

Elementos: ponto (p), linha (l), face (f), curva (curv), curva 2D (curv2), superfície (surf);

Declarações de forma-livre (curvas e superficies): valores de parâmetro (parm), Outer trimming loop (trim), Inner trimming loop (hole), curva especial (scrv), ponto especial (sp), fim de declaração (end);

```
# Cubo

v -1 -1 -1
v 1 -1 -1
v -1 1 -1
v 1 1 -1
v -1 -1 1
v 1 -1 1
v -1 1 1
v 1 1 1
f 1 2 3
f 2 3 4
f 5 6 7
f 6 7 8
f 1 5 7
f 1 3 7
f 1 5 6
f 1 2 6
f 3 7 8
f 3 4 8
f 2 6 8
f 2 8 4
```



Cubo.obj Imagem gerada pelo Mesh Viewer

Fig. 3.1

Conectividade entre superfícies de forma-livre: conectar (con);

Agrupamento: nome de grupo (g), grupo de suavização (s), junção de grupos (mg), nome do objeto (o);

Atributos de visualização/ geração: interpolação Bevel (bevel), interpolaçao de cor (c_interp), dissolver interpolação (d_interp), nível

de detalhe (lod), nome do material (usemtl), biblioteca de material (mtllib), sombreamento (shadow_obj), rastreamento (trace_obj), aproximação técnica de curva (ctech), aproximação técnica de superfície (stech).

Arquivos OBJ não contêm definições de cor das faces, embora possam ser usadas referências que são armazenadas em um arquivo separado de biblioteca. A biblioteca pode ser carregada usando a palavra-chave "mtllib".

Os arquivos OBJ mais comumente encontrados contêm apenas faces poligonais. Para descrever um polígono, o arquivo primeiro descreve cada ponto com a palavra-chave "v", e, em seguida, descreve a face com a palavra-chave "f". A linha de comando de uma face contém as enumerações dos pontos da face, com índices para a lista de pontos, na ordem em que eles ocorrem no arquivo. O seguinte arquivo descreve um triângulo simples:

```
# Simple Wavefront file
v 0.0 0.0 0.0
v 0.0 1.0 0.0
v 1.0 0.0 0.0
f 1 2 3
```

Usaremos esse tipo de arquivo mais simples em nosso trabalho, pois é o suficiente para a construção de malhas triangulares.

Para a visualização dos arquivos, utilizamos o Mesh Viewer, um aplicativo leve e fácil de usar, que exibe modelos tridimensionais (malhas triangulares) a partir de uma variedade de formatos de arquivo. Ele usa OpenGL para gerar os modelos.

O programa nasceu sob a necessidade de exibir rapidamente malhas triangulares. O Mesh Viewer é baseado em uma idéia e uma

implementação antecipada de Craig Robertson. A versão atual foi desenvolvida por Helmut Cantzler e vários colaboradores.

Malhas triangulares podem ser exibidas como texturas mapeadas, sólidos ou wire-frames (aramados). As superfícies normais dos triângulos podem ser exibidas opcionalmente. Modelos carregados podem ser rotacionados, transladados e redimensionados (tudo feito com o mouse). O modelo é iluminado por várias fontes de luz. Pontos de vista podem ser salvos. Imagens do modelo podem ser tomadas. O programa é capaz de calcular o mapa de textura para o modelo 3D. Mais informações e o programa para download pode ser encontrado no site do Mesh Viewer [10].

5.2 Testes com as malhas

Para efetuar os testes, utilizamos o MeshLab v1.1.1 para criar versões simplificadas de algumas malhas de exemplo. A versão simplificada de cada malha consiste numa malha que representa a mesma imagem, mas com menor número de triângulos.

TESTE 1 - cow.obj

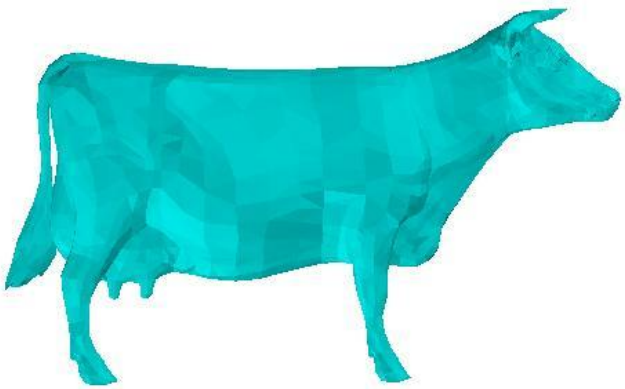


Fig 5.1 – cow.obj

Criamos, a partir da malha de exemplo cow.obj, duas malhas simplificadas: simplified-cow.obj e resimplified-cow.obj

Malha	Vértices	Faces
cow.obj	4583	5804
simplified-cow.obj	1275	2621
resimplified-cow.obj	639	1310

Resultado das avaliações:

Abaixo, mostramos o resultado da avaliação por relação entre círculos inscritos e circunscritos:

```
vba03@macalan:~/tg$ ./meshprisma -aval cow.obj

Nota final da malha: 0.351744, ou 70.348747%
Minima relacao: 0.003055
Maxima relacao: 0.499901
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  19 |  50 |  203 |  220 |  439 |  675 |  894 |  1128 |  1239 |  937 |
Nota:       |   1 |   2 |    3 |    4 |    5 |    6 |    7 |     8 |     9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval simplified-cow.obj

Nota final da malha: 0.373746, ou 74.749121%
Minima relacao: 0.001143
Maxima relacao: 0.499924
```

```
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  29 |  25 |  33 |  86 |  129 |  234 |  333 |  481 |  603 |  668 |
Nota:       |   1 |   2 |   3 |   4 |    5 |    6 |    7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval resimplified-cow.obj

Nota final da malha: 0.374941, ou 74.988194%
Minima relacao: 0.023971
Maxima relacao: 0.499835
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  2 |  5 |  24 |  47 |  77 |  117 |  170 |  246 |  291 |  331 |
Nota:       |  1 |  2 |   3 |   4 |   5 |    6 |    7 |    8 |    9 |   10 |
Execucao concluida
```

A seguir, mostramos o resultado da avaliação utilizando menor ângulo como critério:

```
vba03@macalan:~/tg$ ./meshprisma -aval2 cow.obj

Nota final da malha: 31.717623, ou 52.862706%
Menor angulo: 2.833932
Maior angulo: 59.187428
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  34 |  239 |  472 |  782 |  1022 |  1149 |  946 |  694 |  363 |  103 |
Nota:       |   1 |    2 |    3 |    4 |     5 |     6 |    7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 simplified-cow.obj

Nota final da malha: 35.541745, ou 59.236242%
```

```
Menor angulo: 1.560347
Maior angulo: 59.498461
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  23 |  26 |  120 |  244 |  375 |  501 |  544 |  456 |  240 |  92 |
Nota:       |   1 |   2 |    3 |    4 |    5 |    6 |    7 |    8 |    9 |  10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 resimplified-cow.obj

Nota final da malha: 35.683777, ou 59.472962%
Menor angulo: 6.436143
Maior angulo: 59.043891
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  0 |  8 |  54 |  138 |  208 |  270 |  238 |  221 |  131 |  42 |
Nota:       |  1 |  2 |   3 |    4 |    5 |    6 |    7 |    8 |    9 |  10 |
Execucao concluida
```

Resultado das comparações:

Na seqüência, fizemos comparações entre as malhas.
Para assegurar que os algoritmos funcionam corretamente, testamos as comparações com uma cópia do modelo cow.obj, chamado cow2.obj:

```
vba03@macalan:~/tg$ ./meshprisma -comp cow.obj cow2.obj

Distancia total: 0.000000.
Distancia relativa: 0.000000.
Menor distancia: 0.000000.
Maior distancia: 0.000000
Execucao concluida
```

```
vba03@macalan:~/tg$ ./meshprisma -comp2 cow.obj cow2.obj

Distancia total: 0.000000.
Distancia relativa: 0.000000.
Menor distancia: 0.000000.
Maior distancia: 0.000000
Execucao concluída
```

E, em seguida, a comparação ponto a ponto da malha cow.obj e suas versões simplificadas:

```
vba03@macalan:~/tg$ ./meshprisma -comp cow.obj simplified-cow.obj

Distancia total: 325.156805.
Distancia relativa: 0.070948.
Menor distancia: 0.000000.
Maior distancia: 0.186904
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -comp cow.obj resimplified-cow.obj

Distancia total: 619.614036.
Distancia relativa: 0.135198.
Menor distancia: 0.000000.
Maior distancia: 0.485841
Execucao concluida
```

Utilizando o segundo algoritmo de comparação, obtivemos os seguintes resultados:

```
vba03@macalan:~/tg$ ./meshprisma -comp2 cow.obj simplified-cow.obj

Distancia total: 5.498696.
Distancia relativa: 0.004313.
```

```
Menor distancia: 0.000000.
Maior distancia: 0.095604
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -comp2 cow.obj resimplified-cow.obj

Distancia total: 6.787610.
Distancia relativa: 0.010622.
Menor distancia: 0.000000.
Maior distancia: 0.114305
Execucao concluída
```

TESTE 2 - gourd.obj

Fig 5.2 – gourd.obj

Malha	Vértices	Faces
gourd.obj	4583	5804
simplified-gourd.obj	1275	2621

Resultado das avaliações:

```
vba03@macalan:~/tg$ ./meshprisma -aval gourd.obj

Nota final da malha: 0.375858, ou 75.171522%
Minima relacao: 0.075108
Maxima relacao: 0.499626
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  0 |  3 |  10 |  27 |  33 |  55 |  99 |  101 |  151 |  169 |
Nota:       |  1 |  2 |   3 |   4 |   5 |   6 |   7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval simplified-gourd.obj

Nota final da malha: 0.384137, ou 76.827498%
```

```
Minima relacao: 0.030357
Maxima relacao: 0.497336
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  2 |  1 |  7 |  10 |  14 |  21 |  46 |  45 |  77 |  101 |
Nota:       |  1 |  2 |  3 |   4 |   5 |   6 |   7 |   8 |   9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 gourd.obj

Nota final da malha: 34.138150, ou 56.896917%
Menor angulo: 5.469774
Maior angulo: 58.831295
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  1 |  23 |  38 |  71 |  102 |  114 |  119 |  108 |  49 |  23 |
Nota:       |  1 |   2 |   3 |   4 |    5 |    6 |    7 |    8 |   9 |  10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 simplified-gourd.obj

Nota final da malha: 35.533822, ou 59.223037%
Menor angulo: 7.600212
Maior angulo: 55.605830
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  0 |  11 |  16 |  33 |  40 |  49 |  65 |  68 |  36 |  6 |
Nota:       |  1 |   2 |   3 |   4 |   5 |   6 |   7 |   8 |   9 | 10 |
Execucao concluida
```

Resultado das comparações:

```
vba03@macalan:~/tg$ ./meshprisma -comp gourd.obj simplified-gourd.obj

Distancia total: 17.684907.
```

```
Distancia relativa: 0.054248.
Menor distancia: 0.000000.
Maior distancia: 0.196974
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -comp2 gourd.obj simplified-gourd.obj

Distancia total: 0.621331.
Distancia relativa: 0.003789.
Menor distancia: 0.000000.
Maior distancia: 0.054579
Execucao concluida
```

TESTE 3 - pumpkin.obj

Fig 5.3 – pumpkim.obj

Malha	Vértices	Faces
pumpkin.obj	5002	10000
simplified-pumpkin.obj	2502	5000

Resultado das avaliações:

```
vba03@macalan:~/tg$ ./meshprisma -aval pumpkin.obj

Nota final da malha: 0.344126, ou 68.825214%
Minima relacao: 0.000357
Maxima relacao: 0.499973
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  91 |  212 |  412 |  597 |  878 |  1006 |  1291 |  1562 |  1914 |  2037 |
Nota:       |   1 |    2 |    3 |    4 |    5 |     6 |     7 |     8 |     9 |    10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval simplified-pumpkin.obj

Nota final da malha: 0.353837, ou 70.767357%
```

```
Minima relacao: 0.012064
Maxima relacao: 0.499959
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  26 |  90 |  158 |  264 |  398 |  556 |  604 |  818 |  934 |  1152 |
Nota:       |   1 |   2 |    3 |    4 |    5 |    6 |    7 |    8 |    9 |    10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 pumpkin.obj

Nota final da malha: 32.759322, ou 54.598870%
Menor angulo: 0.738605
Maior angulo: 59.590306
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  27 |  275 |  853 |  1431 |  1647 |  1730 |  1588 |  1332 |  839 |  278 |
Nota:       |   1 |    2 |    3 |     4 |     5 |     6 |     7 |     8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 simplified-pumpkin.obj

Nota final da malha: 33.712701, ou 56.187836%
Menor angulo: 5.140489
Maior angulo: 59.408302
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  2 |  102 |  406 |  653 |  794 |  876 |  835 |  696 |  490 |  146 |
Nota:       |  1 |    2 |    3 |    4 |    5 |    6 |    7 |    8 |    9 |   10 |
Execucao concluida
```

Resultado das comparações:

```
vba03@macalan:~/tg$ ./meshprisma -comp pumpkin.obj simplified-pumpkin.obj

Distancia total: 3924.536615.
```

```
Distancia relativa: 0.784593.
Menor distancia: 0.000000.
Maior distancia: 3.469661
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -comp2 pumpkin.obj simplified-pumpkin.obj

Distancia total: 87.699347.
Distancia relativa: 0.035052.
Menor distancia: 0.000000.
Maior distancia: 0.858162
Execucao concluida
```

TESTE 4 - teapot.obj

Fig 5.4 – teapot.obj

Malha	Vértices	Faces
teapot.obj	3644	6320
simplified-teapot.obj	2348	4607

Resultado das avaliações:

```
vba03@macalan:~/tg$ ./meshprisma -aval teapot.obj

Nota final da malha: 0.273666, ou 54.733150%
Minima relacao: 0.040354
Maxima relacao: 0.485315
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  15 |  581 |  644 |  558 |  440 |  952 |  1231 |  1331 |  537 |  31 |
Nota:       |   1 |    2 |    3 |    4 |    5 |    6 |     7 |     8 |    9 |  10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval simplified-teapot.obj

Nota final da malha: 0.319835, ou 63.966916%
```

```
Minima relacao: 0.000671
Maxima relacao: 0.499977
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  21 |  42 |  131 |  279 |  499 |  781 |  963 |  1127 |  567 |  197 |
Nota:       |   1 |   2 |    3 |    4 |    5 |    6 |    7 |     8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 teapot.obj

Nota final da malha: 22.028048, ou 36.713413%
Menor angulo: 3.353746
Maior angulo: 53.498211
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  483 |  1035 |  685 |  1399 |  1165 |  776 |  562 |  203 |  12 |  0 |
Nota:       |    1 |     2 |    3 |     4 |     5 |    6 |    7 |    8 |   9 | 10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 simplified-teapot.obj

Nota final da malha: 26.660975, ou 44.434958%
Menor angulo: 1.443200
Maior angulo: 59.629771
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  37 |  244 |  678 |  1069 |  887 |  798 |  529 |  272 |  73 |  20 |
Nota:       |   1 |    2 |    3 |     4 |    5 |    6 |    7 |    8 |   9 |  10 |
Execucao concluida
```

Resultado das comparações:

```
vba03@macalan:~/tg$ ./meshprisma -comp teapot.obj simplified-teapot.obj

Distancia total: 41.884823.
```

```
Distancia relativa: 0.011494.
Menor distancia: 0.000000.
Maior distancia: 0.061112
Execucao concluída


vba03@macalan:~/tg$ ./meshprisma -comp2 teapot.obj simplified-teapot.obj

Distancia total: 1.354136.
Distancia relativa: 0.000577.
Menor distancia: 0.000000.
Maior distancia: 0.040717
Execucao concluida
```

TESTE 5 - teddy.obj

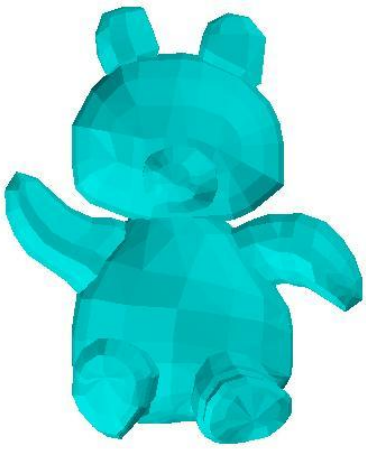


Fig 5.5 – teddy.obj

Malha	Vértices	Faces
teddy.obj	1598	3192
simplified-teddy.obj	514	1024

Resultado das avaliações:

```
vba03@macalan:~/tg$ ./meshprisma -aval teddy.obj

Nota final da malha: 0.353541, ou 70.708100%
Minima relacao: 0.001301
Maxima relacao: 0.499969
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  52 |  69 |  95 |  151 |  225 |  310 |  384 |  523 |  684 |  699 |
Nota:       |   1 |   2 |   3 |    4 |    5 |    6 |    7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval simplified-teddy.obj

Nota final da malha: 0.382248, ou 76.449554%
```

```
Minima relacao: 0.002348
Maxima relacao: 0.499958
Grafico da qualidade dos triangulos, de acordo com a relacao entre ciculos inscritos
e cicunscritos.
Quantidade: |  11 |  6 |  13 |  36 |  35 |  63 |  117 |  206 |  268 |  269 |
Nota:       |   1 |  2 |   3 |   4 |   5 |   6 |    7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 teddy.obj

Nota final da malha: 32.762036, ou 54.603394%
Menor angulo: 0.240663
Maior angulo: 59.547331
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  68 |  138 |  229 |  397 |  469 |  486 |  577 |  415 |  298 |  115 |
Nota:       |   1 |    2 |    3 |    4 |    5 |    6 |    7 |    8 |    9 |   10 |
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -aval2 simplified-teddy.obj

Nota final da malha: 36.315588, ou 60.525980%
Menor angulo: 1.746412
Maior angulo: 59.503413
Grafico da qualidade dos triangulos, de acordo com o menor angulo.
Quantidade: |  7 |  12 |  36 |  73 |  141 |  200 |  250 |  164 |  108 |  33 |
Nota:       |  1 |   2 |   3 |   4 |    5 |    6 |    7 |    8 |    9 |  10 |
Execucao concluida
```

Resultado das comparações:

```
vba03@macalan:~/tg$ ./meshprisma -comp teddy.obj simplified-teddy.obj

Distancia total: 1197.079870.
```

```
Distancia relativa: 0.749111.
Menor distancia: 0.000000.
Maior distancia: 2.652272
Execucao concluida


vba03@macalan:~/tg$ ./meshprisma -comp2 teddy.obj simplified-teddy.obj

Distancia total: 26.074141.
Distancia relativa: 0.050728.
Menor distancia: 0.000000.
Maior distancia: 0.697920
Execucao concluida
```

TESTE 6 - Malhas diferentes

Em seguida, fizemos comparações entre dois modelos diferentes, pumpkin.obj e gourd.obj, a fim de demonstrar as altas distâncias geradas por modelos de objetos diferentes.

```
vba03@macalan:~/tg$ ./meshprisma -comp pumpkin.obj gourd.obj

Distancia total: 550039.053503.
Distancia relativa: 109.963825.
Menor distancia: 75.038202.
Maior distancia: 144.323006
Execucao concluída


vba03@macalan:~/tg$ ./meshprisma -comp2 pumpkin.obj gourd.obj

Distancia total: 24774.184535.
Distancia relativa: 75.994431.
Menor distancia: 75.038202.
Maior distancia: 76.461785
Execucao concluida
```

6. Conclusão

Para efeitos de comparação, utilizamos dois métodos diferentes de avaliação de triângulos e comparação de malhas triangulares.

Os algoritmos de avaliação de malhas triangulares apresentados tiveram ótimos resultados, e verificamos que a avaliação de acordo com o menor ângulo é mais sensível, e diminui a nota de triângulos medianos, ao contrário da avaliação de acordo com a relação entre os círculos inscrito e circunscrito que considera mais triângulos como bons e excelentes. Consideramos a avaliação por menor ângulo melhor, por ser mais rápida.

Após implementar os dois métodos, pudemos perceber que o método de comparação ponto a ponto é mais econômico, do ponto de vista de recursos computacionais, e também mais apropriado para comparação entre malhas de resolução semelhante. Isso ocorre porque se as malhas têm quantidades distintas de triângulos, mesmo que elas representem o mesmo objeto, é inevitável que algum ponto tenha coordenadas diferentes de todos os outros pontos da outra malha, assim gerando uma distância diferente de zero. Para malhas com quantidades parecidas de triângulos, essa pequena distância é aceitável. O método de comparação de malhas ponto a plano é mais preciso, e deve ser utilizado quando as malhas tem resoluções bem diferentes, pois mesmo que os pontos tenham diferentes coordenadas de uma malha para outra, se elas representarem o mesmo objeto, a distância será zero ou muito próxima de zero.

ANEXO I - Algoritmo em pseudocódigo

```
FILE * abre_arquivo(char * nome){ //Caso um nome seja passado para a
funcao, e o arquivo exista, o arquivo e aberto e retornado. Caso
contrario o nome e pedido na saida/entrada padrao, para que entao seja
aberto.
inicializa apontador de arquivo fp
se nome não é nulo
                fp aponta para arquivo nome
fim se
enquanto fp é nulo
                se nome começa com não-nulo
                        exibe mensagem de arquivo inválido
                senão
                        se 2° caractere de nome é maior que 0 ASCII
                                exibe mensagem pedindo nome do arquivo
número recebido
                        senão
                                exibe mensagem pedindo nome do arquivo
                        fim se
                fim se
                escreve buffers
                entra nome
                fp recebe nome
fim enquanto
fim

FILE * abre_malha(char * nomearq, char numeroarq){ //Prepara o nome do
arquivo para a função de abertura de arquivo. Caso esteja em branco,
um flag é colocado no nome do arquivo, seguido do indice do arquivo,
para que a funcao de abertura de arquivo peca o nome na saida/entrada
padrao.
        se não existe nomearq
                nomearq recebe novo string de 50 caracteres
                primeiro caractere de nomearq recebe nulo
                segundo caractere de nomearq recebe número do argumento
        fim se
        retorna abre_arquivo(nomearq)
fim

double quadrado(double a){ //Retorna o quadrado do valor passado por
parametro.
    retorna a*a
fim

void quantidades(FILE * arquivo, int *vertices, int *triangulos){
//Varre o arquivo em busca de vertices e triangulos. Retorna a
quantidade de cada um.
        caractere atual recebe primeiro caractere de arquivo
        contador de vértices recebe 0
        contador de triângulos recebe 0
        enquanto atual não é igual a fim do arquivo
                se atual é #
                        enquanto atual não é igual a nova linha e atual
não é igual a fim do arquivo
                                atual recebe próximo caractere de
arquivo
                        fim enquanto
```

```
                fim se
                se atual é v
                        soma 1 ao contador de vértices
                senão
                        se atual é f
                                soma 1 ao contador de triângulos
                        fim se
                fim se
                atual recebe próximo caractere de arquivo
        fim enquanto
fim

void imprime_histograma(int *histograma, int precisao){ //Coloca um
histograma na saida padrao, em duas linhas: Valores encontrados e
indices.
        inicializa i,j,log1,log2
        imprime mensagem "Quantidade"
        para i recebe 0, i menor que precisão, incrementa i
                imprime formatação e número da coluna
        fim para
        imprime mensagem "Nota"
        para i recebe 0, i menor que precisão, incrementa i
                imprime formatação
                se valor 'i' de histograma é diferente de 0
                        log1 recebe chão de logaritmo base 10 do valor
'i' de histograma
                senão
                        log1 recebe 0
                fim se
                se i é diferente de 0
                        log2 recebe chão de logaritmo base 10 do valor
('i'+1)
                senão
                        log2 recebe 0
                fim se
                para j recebe (log1-log2), j é maior que 0, j é
decrementado
                        imprime formatação
fim para
imprime i+1
        fim para
        imprime formatação
fim


void carrega_malha(FILE * arquivo, double ** vertices, int **
triangulos){ //Percorre o arquivo, colocando todos os valores dos
vertices na tabela de vertices, e, caso uma tabela para triangulos
seja passada, preenche todos os valores dos triangulos. Ignora
comentarios no arquivo.
        inicializa tabela de vértices, tabela de triângulos
        inicializa tabela temporária, auxiliar
        aux recebe próximos dados de arquivo
        enquanto aux não é EOF
                se primeiro caractere de temp é v
                        tabela de vertices recebe vertices
                senão
                        se o primeiro caractere de temp é f
                                se tabela de triângulos é nula
                                        enquanto aux não é \n e aux não
é EOF
```

```
                                                aux recebe próximo
caractere de arquivo
                                        fim enquanto
                                senão
                                        aux recebe dados de arquivo
                                        tabela de triangulos recebe
dados do triangulo
                                fim se
                        senão
                                se primeiro caractere de temp é #
                                        enquanto aux não é \n e aux não
é EOF
                                                aux recebe próximo
caractere de arquivo
                                senão
                                        imprime mensagem de arquivo
inválido
                                fim se
                        fim se
                fim se
        se aux não é EOF
                ler dados de arquivo
        fim se
        fim enquanto
fim

int ** aloca_tabela_int(int x, int y){ //Aloca uma tabela de inteiros,
de acordo com os tamanhos passados por parametro. Tabela usada para
triangulos.
        inicializa tabela, i
        para i=0, i menor que x, incrementa i
                posição i da tabela recebe apontador de tamanho (y *
tamanho em bytes de int)
        fim para
        marca final da tabela com nulo
        retorna tabela
fim

double ** aloca_tabela_double(int x, int y){ //Aloca uma tabela de
doubles, de acordo com os tamanhos passados por parametro. Tabela
usada para vertices.
        inicializa tabela, i
        para i=0, i menor que x, incrementa i
                posição i da tabela recebe apontador de tamanho (y *
tamanho em bytes de double)
        fim para
        marca final da tabela com nulo
        retorna tabela
fim

double distancia(double * p1, double * p2){ //Calcula a distância
entre dois pontos, de acordo com a formula do módulo.
        retorna raiz quadrada de ((p1[0]-p2[0])^2 + (p1[1]-p2[1])^2 +
(p1[2]-p2[2])^2)
fim

double menor_distancia_pontos(double * p1, double * p2, double * p3,
double * ponto_fixo){ //Calcula a distância entre o ponto fixo e os
outros tres pontos. Retorna a menor.
        inicializa menor_dist, dist2, dist3
        menor_dist recebe distancia de p1 a ponto_fixo
```

```
        dist2 recebe distancia de p2 a ponto fixo
        dist3 recebe distancia de p3 a ponto fixo
        se menor_dist é maior que dist2
                menor_dist recebe dist2
        fim se
        se menor_dist é maior que dist3
                menor_dist recebe dist3
        fim se
        retorna menor_dist
fim

double cosseno(double a, double b, double c){ //Calcula pela lei dos
cossenos, o cosseno do angulo oposto ao lado c. Sao passados por
parametro os tamanhos dos lados.
        retorna (quadrado(a)+quadrado(b)-quadrado(c))/(2*a*b)
fim

double seno_metade_angulo(double cossenoa){ //Calcula, pela identidade
da metade-angular, o seno da metade do angulo, baseado no cosseno do
angulo.
    retorna raiz quadrada de ((1-cossenoa)/2);
fim

double seno_de_cosseno(double cosseno){ //De acordo com a formula seno
ao quadrado mais cosseno ao quadrado igual a 1, recebe um cosseno e
retorna um seno. Funciona para fazer o oposto, ja que a formula e a
mesma.
    retorna raiz quadrada de (1-quadrado(cosseno));
fim

double distancia_ponto_plano_com_sinal(double *ponto, double *plano){
//Recebe um vetor de 3 valores, que formam um ponto, e um vetor de 4
valores, que formam um plano (de acordo com a equacao geral do plano,
A, B, C e D). E retorna a distancia entre eles, positiva ou negativa,
de acordo com a posicao relativa entre os dois.

retorna((plano[0]*ponto[0])+(plano[1]*ponto[1])+(plano[2]*ponto[2])+pl
ano[3])/(raiz quadrada de
(quadrado(plano[0])+quadrado(plano[1])+quadrado(plano[2])));
}

double distancia_ponto_plano(double *ponto, double *plano){ //Recebe
um vetor de 3 valores, que formam um ponto, e um vetor de 4 valores,
que formam um plano (de acordo com a equacao geral do plano, A, B, C e
D). E retorna a distancia entre eles, em módulo.
        a recebe distancia_ponto_plano_com_sinal(ponto,plano)
        se a é negativo
                a recebe -a
        fim se
        retorna a
fim

void projecao_ortogonal_ponto_plano(double *ponto, double *plano,
double *retorno){ //Recebe um vetor de 3 valores, que formam um ponto,
e um vetor de 4 valores, que formam um plano (de acordo com a equacao
geral do plano, A, B, C e D). E retorna a projecao ortogonal do ponto
no plano.
        fator recebe (-plano[3]-
(plano[0]*ponto[0]+plano[1]*ponto[1]+plano[2]*ponto[2]))/(quadrado(pla
no[0])+quadrado(plano[1])+quadrado(plano[2]))
        retorno[0] recebe (fator*plano[0])+ponto[0]
```

```
        retorno[1] recebe (fator*plano[1])+ponto[1]
        retorno[2] recebe (fator*plano[2])+ponto[2]
fim

void soma_ponto_vetor(double *ponto, double *vetor, double *retorno){
//Recebe um vetor de 3 valores, que formam um ponto, e um vetor de 3
valores, que formam um vetor no espaço. E retorna a soma do ponto com
o vetor.
    Retorno[0] recebe vetor[0]+ponto[0]
    retorno[1] recebe vetor[1]+ponto[1]
    retorno[2] recebe vetor[2]+ponto[2]
fim

void equacao_do_plano(double *um, double *dois, double *tres, double
*retorno){ //Recebe tres pontos do espaco, e retorna um vetor com os
componentes A, B, C e D da equacao do plano. Calculado segundo o
determinante para equação geral, do livro do paulo boulos.
        inicializa a,b,c
    a recebe ((um[1]-dois[1])*(um[2]-tres[2]))-((um[1]-
tres[1])*(um[2]-dois[2]))
    b recebe ((um[0]-tres[0])*(um[2]-dois[2]))-((um[0]-
dois[0])*(um[2]-tres[2]))
    c recebe ((um[0]-dois[0])*(um[1]-tres[1]))-((um[0]-
tres[0])*(um[1]-dois[1]))
    retorno[3] recebe -(um[0]*a)-(um[1]*b)-(um[2]*c)         //o D da
funcao
    retorno[0] recebe a
    retorno[1] recebe b
    retorno[2] recebe c
fim

void prisma(double *a, double *b, double *c, double *plano, double
**retorno){ //Recebe tres pontos do espaco, equivalentes a vertices de
um triangulo, e o plano formado por aquele triangulo. Retorna a
formula do plano das laterais do prisma, formado com as laterais do
triangulo, e lados perpendiculares a elas.
inicializa vetores de double alinha, blinha, clinha
usa função soma_ponto_vetor(a,plano,alinha)
usa função soma_ponto_vetor(b,plano,blinha)
        usa função soma_ponto_vetor(c,plano,clinha)
        usa função equacao_do_plano(a,b,blinha,retorno[0])
        usa função equacao_do_plano(b,c,clinha,retorno[1])
        usa função equacao_do_plano(c,a,alinha,retorno[2])
fim

int posicao_ponto_prisma(double * ponto, double ** piramide){ //Recebe
um ponto no espaco, e tres planos, que equivalem aos lados de um
prisma de base triangular. O retorno da funcao sinaliza se o ponto
está contido no prisma ou nao
        inicializa doubles dist1,dist2,dist3
        dist1 recebe distancia_ponto_plano_com_sinal(ponto,piramide[0])
        dist2 recebe distancia_ponto_plano_com_sinal(ponto,piramide[1])
        dist3 recebe distancia_ponto_plano_com_sinal(ponto,piramide[2])
//printf("\nponto: %f*%f*%f, distancias: 1:%f, 2:%f,
3:%f",ponto[0],ponto[1],ponto[2],dist1,dist2,dist3)
        se (dist1 é menor ou igual a zero e dist2 é menor ou igual a
zero e dist3 é menor ou igual a zero)
                retorna 1
        fim se
        se (dist1 é maior ou igual a zero e dist2 é maior ou igual a
zero e dist3 é maior ou igual a zero)
```

```
                retorna 1
        fim se
        retorna 0
fim

double relacao_raios(double **vertices, int *triangulo){ //A partir
dos pontos de um triangulo no espaco, a funcao calcula o tamanho dos
lados, e com eles calcula o raio do circulo circunscrito ao triangulo,
e o raio do circulo inscrito. Retorna a razão entre o inscrito e o
circunscrito, nessa ordem.

        inicializa doubles *a=vertices[triangulo[0]],
*b=vertices[triangulo[1]], *c=vertices[triangulo[2]], ab, ac, bc,
cosseno_ab,cosseno_bc,cosseno_ac, raio_circunscrito,
raio_inscrito,variavelteste=1.0
        ab recebe distancia de 'a' a 'b'
        ac recebe distancia de 'a' a 'c'
        bc recebe distancia de 'b' a 'c'
        cosseno_ab recebe o cosseno determinado por ac, bc, ab
        cosseno_ab recebe o cosseno determinado por ac, ab, bc
        cosseno_ab recebe o cosseno determinado por bc, ab, ac
        se cosseno_ab ou cosseno_bc ou cosseno_ac forem muito próximos
de 1
                retorna 0
        fim se
        raio_circunscrito recebe ab/(2*(seno_de_cosseno(cosseno_ab)))
        raio_inscrito recebe
4*raio_circunscrito*seno_metade_angulo(cosseno_ab)*seno_metade_angulo(
cosseno_bc)*seno_metade_angulo(cosseno_ac)
        retorna raio_inscrito/raio_circunscrito
fim

double menor_angulo_triangulo(double **vertices, int *triangulo){ //A
partir dos pontos de um triangulo no espaco, a funcao calcula o
tamanho dos lados, e com eles calcula o valor dos cossenos dos tres
angulos. A partir do maior cosseno, retorna o tamanho do menor angulo.
    inicializa doubles *a=vertices[triangulo[0]],
*b=vertices[triangulo[1]], *c=vertices[triangulo[2]], ab, ac, bc,
maior_cosseno, aux_cosseno, rad2drg=180.0/PI, menor_angulo;

        ab recebe distancia(a,b)
        ac recebe distancia(a,c)
        bc recebe distancia(b,c)
        maior_cosseno recebe cosseno(ac,bc,ab)
        aux_cosseno recebe cosseno(ab,ac,bc)
        se aux_cosseno é maior que maior_cosseno
                maior_cosseno recebe aux_cosseno
        fim se
        aux_cosseno recebe cosseno(ab,bc,ac)
        se aux_cosseno é maior que maior_cosseno
                maior_cosseno recebe aux_cosseno
        fim se
        menor_angulo recebe acosf(maior_cosseno)*rad2drg
        retorna menor_angulo
fim

int * relacao_circulos(double **vertices,int **triangulos, int
precisao, int saidapadrao){ //Calcula as razoes entre o circulo
inscrito e o circulo circunscrito de uma lista de triangulos. O raio
do circulo inscricrito, dividido pelo raio do circulo circunscrito,
varia entre quase zero e meio, de acordo com a equilaridade do
```

```
triangulo. A funcao ou joga todas as razoes na saida padrao, ou
calcula um histograma de incidencia de razoes, de precisao definida
pelo usuario. Também registra a menor e a maior razão, e calcula uma
media entre todas, determinada nota final. Retorna o histograma, para
que seja colocado na saida padrao
    inicializa ints i,*histograma
    inicializa doubles
relacao,nota_final=0,minima_relacao=HUGE_VAL,maxima_relacao=0
    se saidapadrao é falso
        histograma recebe apontador de int com tamanho precisão *
tamanho de int
        para i recebe 0, i menor que precisao, incrementa i
               posição i de histograma[] recebe 0
        fim para
    fim se
    para i recebe 0, posição i de triangulos[] não é nulo, incrementa
i
        relacao recebe relacao_raios(vertices,posição i de
triangulos[]);
        se (relacao<minima_relacao)
               minima_relacao=relacao
        fim se
        se (relacao>maxima_relacao)
               maxima_relacao=relacao
        fim se
        se saidapadrao é verdadeiro
            imprime relacao
        se não
            se relacao é igual a 0
                incrementa posição 0 de histograma
            se não
                incrementa a posição apropriada do histograma
            fim se
            nota_final recebe nota_final+relacao
        fim se
    fim para
    se saidapadrao é falso
        nota_final recebe nota_final/i
        imprime nota final da malha, minima relação, máxima relação
        imprime título do histograma
    fim se
    retorna histograma
fim

int * menores_angulos(double **vertices,int **triangulos, int
precisao, int saidapadrao){ //Calcula os menores angulos de uma lista
de triangulos. O menor angulo de um triangulo varia entre quase zero e
sessenta, de acordo com a equilaridade do triangulo. A funcao ou joga
todas os angulos na saida padrao, ou calcula um histograma de
incidencia dos angulos, de precisao definida pelo usuario. Também
registra o menor e o maior "menor angulo", e calcula uma media entre
todas, determinada nota final.
        inicializa ints i, *histograma
        inicializa
menor_angulo,nota_final=0,minimo_angulo=HUGE_VAL,maximo_angulo=0
        se saidapadrao é falso
               histograma recebe o tamanho de int*precisão
               para i recebe 0, i menor que precisao, incrementa i
                       indice i de histograma[] recebe 0
               fim para
        fim se
```

```
        para i recebe 0, indice i de triangulos[] é diferente de nulo,
incrementa i
                menor_angulo recebe resultado de
menor_angulo_triangulo(vertices, indice i de triangulos[])
                se menor_angulo é menor que minimo_angulo
                        minimo_angulo recebe menor_angulo
                fim se
                se menor_angulo é maior que maximo_angulo
                        maximo_angulo recebe menor_angulo
                fim se
                se saidapadrao é verdadeiro
                        imprime o valor de menor_angulo
                se não
                        se menor angulo é menor ou igual a 0
                                incrementa indice 0 do histograma
                        se não
                                se menor_angulo é maior ou igual a 60
                                        incrementa indice precisao-1
(ultimo indice) do histograma
                                se não
                                        incrementa indice teto de
((menor_angulo/60*precisao) - 1)
                                fim se
                        fim se
                nota_final recebe nota_final+menor_angulo
        fim se
        fim para
        se saidapadrao é falso
                nota_final recebe nota_final/i
                imprime nota final da malha, menor angulo, maior angulo
                imprime texto do histograma
        fim se
        retorna histograma
fim

double compara_malhas(double ** vertices1, double ** vertices2, int
saidapadrao){ //Calcula a distancia total, distancia relativa, menor
distancia e maior distancia entre duas malhas de triangulos, baseado
na menor distancia entre os pontos de duas malhas. Cada ponto de uma
malha é comparado com todos os da outra. Nao leva em conta a
triangulacao, apenas os pontos.
        inicializa ints i,j
    inicializa doubles *
aux,menor_distancia,distancia_atual,minima_distancia=HUGE_VAL,maxima_d
istancia=0
    inicializa double distancia_total=0

        para i recebe 0, indice i de vertices1 não é nulo, incrementa i
                para j recebe 0 e menor_distancia recebe maximo valor
possível, indice j de vertices2 é não-nulo e menor_distancia é
diferente de 0, incrementa j
                        distancia_atual recebe resultado de distancia
entre índice i de vertices1 e índice j de vértices2
                        se distancia_atual<menor_distancia
                                menor_distancia recebe distancia_atual
                        fim se
                fim para
                se saidapadrao é verdadeiro
                        imprime menor_distancia
                se não
```

```
                        distancia_total recebe
distancia_total+menor_distancia
                fim se
                se menor_distancia é menor que minima_distancia,
minima_distancia recebe menor_distancia
                se menor_distancia é maior que maxima_distancia,
maxima_distancia recebe menor_distancia
        fim para
        se saidapadrão é falso
                imprime distancia total, distancia relativa, menor
distancia, maior distancia
        fim se
        retorna distancia_total/i
fim

double compara_planos(double ** vertices1, double ** vertices2, int **
triangulos1, int qtdtriangulos1, int saidapadrao){ //Calcula a
distancia total, distancia relativa, menor distancia e maior distancia
entre duas malhas de triangulos, baseado na menor distancia entre os
pontos de uma malha e os planos formados pelos triangulos de outra
malha. A funcao comeca alocando uma tabela de planos, e calcula a
formula do plano para cada triangulo de uma das malhas. Para cada
plano é calculado a distancia para todos os pontos da outra malha (ou
até que seja encontrada uma distancia igual a zero). Nao leva em conta
os limites dos triangulos, cada plano tem tamanho infinito.
        inicializa ints i,j,k
    inicializa doubles *
aux,menor_distancia,distancia_atual,minima_distancia=HUGE_VAL,maxima_d
istancia=0,**planos,**prisma_atual
    inicializa double distancia_total=0
    planos recebe o resultado de aloca_tabela_double(qtdtriangulos1,4)
    prisma_atual recebe o resultado de aloca_tabela_double(3,4)
    para i recebe 0, i menor que qtdtriangulos1, incrementa i

        equacao_do_plano(vertices1[triangulos1[i][0]],vertices1[triangu
los1[i][1]],vertices1[triangulos1[i][2]],planos[i])
        para i recebe 0, indice i de vertices2 é não-nulo, incrementa i
                para j recebe 0 e menor_distancia recebe maior valor
possivel, indice j de planos é não-nulo e menor_distancia é diferente
de 0, incrementa j

        prisma(vertices1[triangulos1[j][0]],vertices1[triangulos1[j][1]
],vertices1[triangulos1[j][2]],planos[j],prisma_atual)
                        se resultado de posicao_ponto_prisma(indice i
de vertices2, prisma_atual é verdadeiro
                                distancia_atual recebe resultado de
distancia_ponto_plano(indice i de vertices2, indice j de planos)
                        se não
                                distancia_atual recebe resultado de
menor_distancia_pontos(vertices1[triangulos1[j][0]],vertices1[triangul
os1[j][1]],vertices1[triangulos1[j][2]],vertices2[i])
                        fim se
                        se distancia_atual é menor que menor_distancia
                                menor_distancia recebe distancia_atual
                        fim se
                fim para
                se saidapadrao é verdadeiro
                        imprime menor_distancia
                se não
                        distancia_total recebe distancia_total +
menor_distancia
```

```
                fim se
                se menor distancia é menor que minima_distancia
                        minima_distancia recebe menor_distancia
                fim se
                se menor distancia é maior que maxima_distancia
                        maxima_distancia recebe menor_distancia
                fim se
        fim para
        se saidapadrao é falso
                imprime distancia total, distancia relativa, menor
distancia, maior distancia
        fim se
        retorna distancia_total/i
fim

void avalia_circulos(char * nomearq, int saidapadrao, int precisao){
//Funcao que cria as condicoes para que a funcao relacao circulos seja
executada:
    inicializa apontador de arquivo
    inicializa ints qtdvertices,
qtdtriangulos,**triangulos,*histograma
    inicializa double **vertices
    abre o arquivo
    calcula as quantidades de vertices e triangulos
    volta ao inicio do arquivo
    aloca a tabela de vertices
    aloca a tabela de triangulos
    extrai as informacoes do arquivo, e coloca nas respectivas tabelas
    chama a função que calcula a razao entre os circulos, e ou coloca
o resultado no histograma, ou coloca os resultados na saida padrao
    se saidapadrao é falso
        imprime o histograma
    fim se
fim

void avalia_angulos(char * nomearq, int saidapadrao, int precisao){
        inicializa apontador de arquivo
        inicializa ints qtdvertices,
qtdtriangulos,**triangulos,*histograma
        inicializa double **vertices;
        abre arquivo
        calcula as quantidades de vértices e triângulos
        volta ao início do arquivo
        aloca tabela de vértices
        aloca tabela de triângulos
        extrai as informações de vértices e triângulos do arquivo para
as respectivas tabelas
        cria o histograma com os menores angulos
        se saidapadrão é falso
                imprime histograma com a precisão determinada
        fim se
fim

void comparacao(char * entrada1, char * entrada2, int saidapadrao){
        inicializa apontadores de arquivo
        inicializa ints
qtdvertices1,qtdvertices2,qtdtriangulos1,qtdtriangulos2
        inicializa doubles double **vertices1,**vertices2,nota_final
        abre arquivo1
        calcula as quantidades de vértices e triângulos
        volta ao início do arquivo
```

```
        abre arquivo2
        calcula as quantidades de vértices e triângulos
        volta ao início do arquivo
        vertices1 recebe quantidade de vértices do arquivo 1
        vertices2 recebe quantidade de vértices do arquivo 2
        carrega vértices do arquivo1
        carrega vértices do arquivo2
        nota_final recebe o resultado de compara_malhas() entre
vertices1 e vertices2
fim

void comparacao_planos(char * entrada1, char * entrada2, int
saidapadrao){
        inicializa apontadores de arquivo arquivo1, arquivo2
        inicializa ints
qtdvertices1,qtdvertices2,qtdtriangulos1,qtdtriangulos2,**triangulos1
    inicializa doubles **vertices1,**vertices2,nota_final
        arquivo1 recebe entrada1
        calcula as quantidades de vertices e triangulos de arquivo1
        volta ao início de arquivo1
        arquivo2 recebe entrada2
        calcula as quantidades de vertices e triangulos de arquivo2
        volta ao início de arquivo2
        vertices1 recebe resultado de
aloca_tabela_double(qtdvertices1,3)
        vertices2 recebe resultado de
aloca_tabela_double(qtdvertices2,3)
        triangulos1 recebe resultado de
aloca_tabela_int(qtdtriangulos1,3)
    executa carrega_malha(arquivo1,vertices1,triangulos1)
    executa carrega_malha(arquivo2,vertices2,NULL)
        nota_final recebe resultado de
compara_planos(vertices1,vertices2,triangulos1,qtdtriangulos1,saidapad
rao)

int parametros(int argc, char ** args, char ** entrada1, char **
entrada2, int *saidapadrao, int *precisao){
        inicializa int i, funcao=0, ientrada=0
    inicializa char *** entradas;
        funcao recebe o argumento recebido da linha de comando
        se argumento de função é inválido
                imprime mensagem de erro
                interrompe execução do programa com valor de retorno -1
        fim se
        retorna funcao
fim

int main(int argc, char ** args){
    inicializa ints i,funcao,saidapadrao=0,precisao=10
    inicializa char **entrada1=(char**)malloc(sizeof(char*)),
**entrada2=(char**)malloc(sizeof(char*))
    inicializa apontador *entrada1=NULL
    inicializa apontador *entrada2=NULL

        funcao recebe resultado de
parametros(argc,args,entrada1,entrada2,&saidapadrao,&precisao)

    switch(funcao)
        caso 1 :
                        avalia_circulos(*entrada1, saidapadrao,
precisao)
```

```
                        break
        caso 2 :
                        avalia_angulos(*entrada1, saidapadrao,
precisao)
                        break
        caso 3 :
                        comparacao(*entrada1, *entrada2, saidapadrao)
                        break
        caso 4 :
                        comparacao_planos(*entrada1, *entrada2,
saidapadrao)
                        break
    fim switch

    se saidapadrao é falso
                imprime Execução concluida
        fim se
fim
```

ANEXO II - Descrição das Funções

Arquivos utilizados: meshprisma.c, Makefile, readme

O programa ainda conta com um parâmetro de saída padrão, *-sp*, que faz com que os dados da avaliação ou comparação sejam jogados na saída padrão em vez de mostrá-los resumidamente em estatísticas. Na avaliação utilizando as medidas dos círculos inscritos e circunscritos, o parâmetro mostra as razões entre os raios dos círculos inscrito e circunscrito de cada triângulo, sendo no máximo 0,5, separadas por espaço. Na avaliação de menor ângulo de um triângulo, ele mostra o menor ângulo de cada triângulo em graus, sendo no máximo 60°. Quanto maior esse ângulo, melhor é o triângulo. Na comparação ponto a ponto, o parâmetro mostra as distâncias entre cada ponto e o ponto mais próximo a ele na outra malha, separadas por espaços. Na comparação entre ponto e plano, mostra a distância de cada plano para o ponto mais próximo na outra malha.

meshprisma.c

Contém a função inicial do programa, bem como as funções responsáveis pela leitura dos vértices e dos triângulos e também as funções responsáveis pela medição e comparação de malhas.

FILE * abre_arquivo(char * nome): Se um nome de arquivo válido for passado para a função, o arquivo é aberto. Caso contrário, o nome do arquivo será solicitado na saída/entrada padrão para que então seja aberto.

FILE * abre_malha(char * nomearq, char numeroarq): Preparação para abrir o arquivo. Caso o nome esteja em branco, um

flag é colocado no nome do arquivo, seguido do índice do arquivo, para que a função de abertura de arquivo solicite o nome na saída/entrada padrão.

double quadrado(double a): Retorna o quadrado do valor passado por parâmetro.

void quantidades(FILE * arquivo, int *vertices, int *triangulos): Varre o arquivo em busca de vértices e triângulos. Retorna a quantidade de cada um.

void imprime_histograma(int *histograma, int precisao): Imprime um histograma na saída padrão, com duas linhas: valores encontrados e índices. Escreve a nota precedida por espaços: número de casas decimais da "quantidade", menos o número de casas decimais da nota, mais dois, para garantir o alinhamento das duas linhas.

void carrega_malha(FILE * arquivo, double ** vertices, int ** triangulos): Percorre o arquivo, colocando todos os valores dos vértices na tabela de vértices, e, caso uma tabela para triângulos seja passada, preenche todos os valores dos triângulos. Ignora comentários no arquivo.

int ** aloca_tabela_int(int x, int y): Aloca uma tabela de inteiros, de acordo com os tamanhos passados por parâmetro. Essa tabela é usada para triângulos.

double ** aloca_tabela_double(int x, int y): Aloca uma tabela de doubles, de acordo com os tamanhos passados por parâmetro. Essa tabela é usada para vértices.

double distancia(double * p1, double * p2): Calcula a distância entre dois pontos, de acordo com a fórmula do módulo.[fórmula]

double menor_distancia_pontos(double * p1, double * p2, double * p3, double * ponto_fixo): Calcula a distância entre o ponto fixo e os outros três pontos. Retorna a menor distância.

double cosseno(double a, double b, double c): Calcula pela lei dos cossenos [lei], o cosseno do ângulo oposto ao lado c. Os tamanhos dos lados são passados como parâmetro.

double seno_metade_angulo(double cossenoa): Calcula, pela identidade da metade-angular [identidade], o seno da metade do ângulo, baseado no cosseno do ângulo.

double seno_de_cosseno(double cosseno): De acordo com a fórmula [seno ao quadrado mais cosseno ao quadrado igual a 1], recebe um cosseno e retorna um seno. Funciona para fazer o oposto, ja que a fórmula é a mesma.

double distancia_ponto_plano_com_sinal(double *ponto, double *plano): Recebe como parâmetros um vetor de 3 valores, que formam um ponto, e um vetor de 4 valores, que formam um plano (de acordo com a equação geral do plano, A, B, C e D). E retorna a distância entre eles, positiva ou negativa, de acordo com a posição relativa entre os dois.

double distancia_ponto_plano(double *ponto, double *plano): Recebe como parâmetros um vetor de 3 valores, que formam um ponto, e um vetor de 4 valores, que formam um plano (de acordo com a equação geral do plano, A, B, C e D). Retorna a distância entre eles, em módulo.

void projecao_ortogonal_ponto_plano(double *ponto, double *plano, double *retorno): Recebe como parâmetros um vetor de 3 valores, que formam um ponto, e um vetor de 4 valores, que formam um plano (de acordo com a equação geral do plano, A, B, C e D). Retorna a projeção ortogonal do ponto no plano.

void soma_ponto_vetor(double *ponto, double *vetor, double *retorno): Recebe como parâmetros um vetor de 3 valores, que formam um ponto, e um vetor de 3 valores, que formam um vetor no espaço. Retorna a soma do ponto com o vetor.

void equacao_do_plano(double *um, double *dois, double *tres, double *retorno): Recebe como parâmetros três pontos do espaço. Retorna um vetor com os componentes A, B, C e D da equação do plano. É calculado segundo o determinante para equação geral [7].

void prisma(double *a, double *b, double *c, double *plano, double **retorno): Recebe como parâmetros três pontos do espaço, equivalentes aos vértices de um triângulo, e o plano formado por aquele triângulo. Retorna a fórmula do plano das laterais do prisma, formado com as laterais do triângulo, e os lados perpendiculares a elas.

int posicao_ponto_prisma(double * ponto, double ** piramide): Recebe como parâmetros um ponto no espaço, e três planos, que equivalem aos lados de um prisma de base triangular. O retorno da função sinaliza se o ponto está contido no prisma ou não.

double relacao_raios(double **vertices, int *triangulo): A partir dos pontos de um triângulo no espaço, a função calcula o tamanho

dos lados, e com eles calcula o raio do círculo circunscrito ao triângulo, e o raio do círculo inscrito. Retorna a razão entre o inscrito e o circunscrito, nessa ordem.

double menor_angulo_triangulo(double **vertices, int *triangulo): A partir dos pontos de um triângulo no espaço, a função calcula o tamanho dos lados, e com eles calcula o valor dos cossenos dos três ângulos. A partir do maior cosseno, retorna o tamanho do menor ângulo.

int * relacao_circulos(double **vertices,int **triangulos, int precisao, int saidapadrao): Calcula as razões entre o círculo inscrito e o círculo circunscrito de uma lista de triângulos. O raio do círculo inscrito, dividido pelo raio do círculo circunscrito, varia entre quase zero e meio, de acordo com os ângulos do triângulo. A função joga todas as razões na saída padrão, ou calcula um histograma de incidência de razões, de precisão definida pelo usuário. Também registra a menor e a maior razão, e calcula uma média entre todas, determinada nota final. Retorna o histograma, para que seja colocado na saída padrão.

int * menores_angulos(double **vertices,int **triangulos, int precisao, int saidapadrao): Calcula os menores ângulos de uma lista de triângulos. O menor ângulo de um triângulo varia entre quase zero e sessenta, de acordo com os ângulos do triângulo. A função joga todos os ângulos na saída padrão, ou calcula um histograma de incidência dos ângulos, de precisão definida pelo usuário. Também registra o menor e o maior "menor ângulo", e calcula uma média entre todos, determinada nota final.

double compara_malhas(double ** vertices1, double ** vertices2, int saidapadrao): Calcula a distância total, distância

relativa, menor distância e maior distância entre duas malhas de triângulos, baseado na menor distância entre os pontos de duas malhas. Cada ponto de uma malha é comparado com todos os da outra. Não leva em conta a triangulação [definir no inicio], apenas os pontos.

double compara_planos(double ** vertices1, double ** vertices2, int ** triangulos1, int qtdtriangulos1, int saidapadrao): Calcula a distância total, distância relativa, menor distância e maior distância entre duas malhas de triângulos, baseado na menor distância entre os pontos de uma malha e os planos formados pelos triângulos de outra malha. A função começa alocando uma tabela de planos, e calcula a fórmula do plano para cada triângulo de uma das malhas. Para cada plano é calculado a distância para todos os pontos da outra malha (ou até que seja encontrada uma distância igual a zero). Não leva em conta os limites dos triângulos, cada plano tem tamanho infinito.

void avalia_circulos(char * nomearq, int saidapadrao, int precisao): Função que cria as condições para que a função relacao_circulos seja executada. Chama a função que calcula a razão entre os círculos, e coloca o resultado no histograma (se solicitado), ou na saída padrão.

void avalia_angulos(char * nomearq, int saidapadrao, int precisao): Função que cria as condições para que a função menores_angulos seja executada. Chama a função que calcula os menores ângulos de uma lista de triângulos, e coloca o resultado no histograma (se solicitado), ou na saída padrão.

void comparacao(char * entrada1, char * entrada2, int saidapadrao): Função que cria as condições para que a função

compara_malhas seja executada. Chama a função que calcula a distância total, distância relativa, menor distância e maior distância entre duas malhas de triângulos, e coloca o resultado no histograma (se solicitado), ou na saída padrão.

void comparacao_planos(char * entrada1, char * entrada2, int saidapadrao): Função que cria as condições para que a função compara_planos seja executada. Chama a função que calcula a distância total, distância relativa, menor distância e maior distância entre duas malhas de triângulos, e coloca o resultado no histograma (se solicitado), ou na saída padrão.

int parametros(int argc, char ** args, char ** entrada1, char ** entrada2, int *saidapadrao, int *precisão).

Makefile

Contém um script de compilação do programa. Para a sua execução basta executar o comando make no diretório dos arquivos do programa.

readme

Contém uma breve descrição de como compilar e de como executar o programa.

Referências

[1] O. M. van Kaick; H. Pedrini, **Métricas para simplificação de malhas triangulares. –** Curitiba: Universidade Federal do Paraná, 2005.

[2] Eduardo Sant'Ana da Silva. **Compressão Eficiente da Geometria de Malhas Triangulares.** - Curitiba: Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, 2005.

[3] Zeeman, C.E.. **Uma introdução informal à topologia das superfícies.** – Rio de Janeiro: Instituto de Matemática Pura e Aplicada, [1975?].

[4] O.M. van Kaick; H. Pedrini, **Estudo Comparativo de Métodos de Compressão de Modelos de Terrenos Digitais através de Superfícies Triangulares**. – Curitiba: Proceedings of III Colóquio Brasileiro de Ciências Geodésicas, 2003.

[5] Ricardo Luís Barbosa; Messias Meneguette Jr.; João Fernando Custódio da Silva; Rodrigo Bezerra de Araújo Gallis. **Geração de modelo digital do terreno utilizando a triangulação de Delaunay e Thin Plate Spline. –** Curitiba: Anais do III Colóquio Brasileiro de Ciências Geodésicas, 2003.

[6] **Geometria Espacial: Elementos de Geometria Espacial**. Disponível em:
<http://pessoal.sercomtel.com.br/matematica/geometria/element/element.htm>
Acesso em: 06 jul. 2008.

[7] Paulo Boulos; Ivan de Camargo. **Geometria Analítica: Um tratamento vetorial.** – São Paulo: McGraw-Hill, 1986.

[8] Marcelo Walter. **SIVOP - Sistema Visualizador de Objetos Poliédricos.** – UFPE. Disponível em:
<http://www.cin.ufpe.br/~marcelow/Marcelow/sivop.html>
Acesso em: 06 jul. 2008.

[9] **Wavefront OBJ File Format Summary.** Disponível em:
<http://www.fileformat.info/format/wavefrontobj/>
Acesso em: 06 jul. 2008.

[10] **Mesh Viewer.** Disponível em:
<http://mview.sourceforge.net/>
Acesso em: 06 jul. 2008.

[11] Guilherme M. Magalhães; Angelo Pássaro; Nancy Mieko Abe. **Geração de malha de Delaunay orientada a objetos. -** São José dos Campos, SP: Anais do Worcomp'2000 – Workshop de Computação, 2000.

[12] André Luiz Pires Guedes. **Introdução à Geometria Computacional.** – UFPR. Disponível em:
<http://www.inf.ufpr.br/andre/geom/node8.html>
Acesso em: 06 jul. 2008.

[13] Silva, S. Madeira; J. Santos, B.S. **POLYMECO — A Polygonal Mesh Comparison Tool. -** Washington, DC, USA: IEEE Computer Society , 2005.

[14] Ricardo Luís Lachi; Heloísa Vieira da Rocha. **Aspectos básicos de clustering: conceitos e técnicas**. – UNICAMP, 2005.

[15] Raphael de Oliveira Borges; Rosane Amaral Alves da Silva; Selma Simões de Castro. **Utilização da classificação por distância euclidiana no mapeamento dos focos de arenização no setor sul da alta bacia do Rio Araguaia.** – Florianópolis: Anais XIII Simpósio Brasileiro de Sensoriamento Remoto, 2007.

[16] **Math Open Reference.** Disponível em:
< http://www.mathopenref.com/>
Acesso em: 06 jul. 2008.

[17] **The GTS Library.** Disponível em:
<http://gts.sourceforge.net/samples.html>
Acesso em: 06 jul. 2008.