

UEM – UNIVERSIDADE ESTADUAL DE MARINGÁ
DIN – DEPARTAMENTO DE INFORMÁTICA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS PARA WEB

MODELAGEM DE SUPERFÍCIE NA WEB UTILIZANDO
TRIANGULAÇÃO DE DELAUNAY

CÉSAR ALBERTO DA SILVA

Maringá – PR
2005

UEM – UNIVERSIDADE ESTADUAL DE MARINGÁ
DIN – DEPARTAMENTO DE INFORMÁTICA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS PARA WEB

MODELAGEM DE SUPERFÍCIE NA WEB UTILIZANDO
TRIANGULAÇÃO DE DELAUNAY

CÉSAR ALBERTO DA SILVA

Trabalho monográfico apresentado no curso de especialização, Desenvolvimento de Sistemas para Web, como requisito parcial para sua conclusão.

Orientador:
Prof. Dr. Dante Alves Medeiros Filho.

Maringá - PR
2005

DEDICATÓRIA

Dedico este trabalho à minha mãe Ivete, que nunca mediu esforços e esteve sempre presente em todos os momentos de sua realização. Ao meu pai Alberto, pelo carinho e confiança, e à minha namorada Soellyn pela compreensão e paciência.

AGRADECIMENTOS

À minha família que, em todos os momentos de realização desta pesquisa, esteve presente.

Agradecimentos também ao professor orientador, Dr. Dante que, na rigidez de seus ensinamentos, fez aprimorar meus conhecimentos.

Aos meus amigos, pelo companheirismo e muitos momentos de alegria compartilhados.

EPÍGRAFE

"[...] nada é fixo para aquele que alternadamente pensa e sonha [...]"

Gaston Bachelard

"A vida só pode ser compreendida olhando-se para trás; mas só pode ser vivida olhando-se para a frente."

Soren Kierkegaard

"O que prevemos raramente ocorre; o que menos esperamos geralmente acontece."

Benjamin Disraeli

SILVA, César Alberto. **Modelagem de Superfície na Web utilizando Triangulação de Delaunay**. Maringá: UEM – Universidade Estadual de Maringá, 2005. (Monografia de Especialização).

Orientador:

Prof. Dr. Dante Alves Medeiros Filho

RESUMO

Neste trabalho foi implementada a Triangulação de Delaunay no plano de pontos não uniformemente distribuídos no espaço, através de uma estrutura dinâmica de dados, que permite a interação do usuário com os pontos, para realizar operações de mudança e exclusão, permitindo a alteração da malha triangular em tempo real. A Triangulação de Delaunay, juntamente com o critério do círculo, otimiza a triangulação gerando triângulos o mais equiláteros possíveis. A interpolação utilizada foi a Interpolação Linear. A interpolação de dados não uniformemente distribuídos tem aplicações em várias áreas do conhecimento, como por exemplo, em Computação Gráfica, Geração de Modelo Digital do Terreno, escoamento de Fluidos entre outros. Para que a interpolação possa ser realizada, é necessário organizar os dados em malhas e a Triangulação de Delaunay é muito utilizada para esse fim.

SILVA, César Alberto. **Modeling of Surface in the Web using Triangulation of Delaunay**. Maringá: UEM – Universidade Estadual de Maringá, 2005. (Monograph of Graduation).

Advisor:

Dante Alves Medeiros Filho, Phd.

ABSTRACT

In this work the Triangulation of Delaunay in the plan of points not uniformly distributed in the space was implemented, through a dynamic structure of data, that allows the interaction of the user with the points, to carry through operations of change and exclusion, allowing the alteration of the triangular mesh in real time. The Triangulation of Delaunay, together with the criterion of the circle, optimizes the triangulation generating the possible most equilateral triangles. The used interpolation was the Linear Interpolation. The interpolation of data not uniformly distributed has applications in some areas of the knowledge, as for example, in Graphical Computation, Generation of Digital Model of the Land, Fluid Draining among others. So that the interpolation can be carried through, it is necessary to organize the data in meshes and the Triangulation of Delaunay very is used for this end.

SUMÁRIO

LISTA DE FIGURAS	1
LISTA DE ABREVIATURAS	2
1. Introdução	3
1.1. Objetivo Geral	4
1.2. Objetivo Específico.....	4
1.3. Definição do Problema	5
1.4. Justificativa.....	6
1.5. Limitações da pesquisa.....	7
1.6. Obtenção das informações da superfície	7
2. Triangulação.....	8
3. Critério do círculo	11
4. Modelo Digital de Terreno	16
5. Interpolação.....	18
5.1. Interpolação Linear.....	19
6. Visualização da Superfície	21
7. Resultados	23
8. Considerações Finais.....	31
9. Bibliografia.....	32
Apêndice.....	33

LISTA DE FIGURAS

Figura 1	- Critério do Círculo	09
Figura 2	- Único vértice da Tesselação de Dirichlet	10
Figura 3	- Critério do Círculo	11
Figura 4	- Triangulação de Delaunay	13
Figura 5	- Triângulo que não é Delaunay	14
Figura 6	- Aresta Delaunay	14
Figura 7	- Sistema de coordenadas do mundo	20
Figura 8	- Sistema de coordenadas do <i>viewport</i>	20
Figura 9	- Tela Inicial do Software	23
Figura 10	- Triangulação de Delaunay	24
Figura 11	- Triangulação de Delaunay com a remoção de um ponto	25
Figura 12	- Triangulação de Delaunay com a remoção de dois pontos	26
Figura 13	- Triangulação de Delaunay com a inserção de vários pontos aleatórios	27
Figura 14	- Malha Regular 30x30	28
Figura 15	- Superfície gera pela Interpolação Linear	29

LISTA DE ABREVIATURAS

- SIG - Sistemas de Informação Geográfica
- MDT - Modelagem de Terreno Digital

1. Introdução

Para o planejamento das obras de engenharia, é necessário que se tenha um mapa ou planta topográfica, que represente a área de interesse ao projeto, com informações adequadas e atualizadas. A representação da superfície topográfica de forma adequada permite a obtenção de diversas informações, tais como: a distância vertical entre pontos, a inclinação de taludes, a construção de perfis, intervisibilidade entre pontos, elementos para o cálculo de áreas e volumes, entre outras.

Com a criação das técnicas de modelagem digital do terreno, tornou-se possível solucionar diretamente todos esses problemas, através de um modelo numérico representativo do terreno que pode ser trabalhado para se obter as informações desejadas.

A modelagem digital de terreno tem facilitado em muito os trabalhos cartográficos, contudo, há necessidade de se verificar a qualidade dos produtos obtidos, através da avaliação por métodos estatísticos. Como a quantidade de pontos amostrados é insuficiente para uma boa representação do terreno, é necessário interpolar pontos para gerar a superfície.

Uma forma muito útil de se agrupar os pontos para a posterior interpolação é a triangulação desses pontos no plano e a triangulação mais utilizada é a de Delaunay.

A malha triangular agrupa os pontos irregularmente espaçados e a superfície é gerada com a criação de uma outra malha (regular) sobre essa, onde os pontos z são calculados com uma função interpoladora $z = f(x, y)$.

Existe uma grande variedade de funções interpoladoras como por exemplo: Splines, Thin Plate Splines, Krigagem, superfícies de tendência etc. Nesse trabalho, foi implementada uma Interpolação Linear.

A triangulação é feita com o algoritmo incremental e é utilizada uma estrutura de dados dinâmica que permite uma flexibilidade na interação do usuário com a malha triangular, como inclusão, remoção e alteração dos pontos.

Com a malha triangular o usuário define o tamanho da malha regular e é realizada a interpolação para os pontos que pertençam ao interior da triangulação e finalmente uma visualização da superfície é gerada.

A ferramenta foi desenvolvida para web em razão do grande avanço da internet, por isso facilita o compartilhamento das informações em tempo real independentemente da localização.

1.1. Objetivo Geral

Desenvolver uma ferramenta para modelagem digital de terreno via web.

1.2. Objetivo Específico

- Implementar a Triangulação de Delaunay no plano de pontos não uniformemente distribuídos.
- Criar uma estrutura de dados dinâmica para armazenamento dos pontos.

- Permitir a interação do usuário com os pontos iniciais para realizar operações de mudança e exclusão de tais pontos com a alteração da malha triangular em tempo real.
- Interpolar a superfície com pontos interiores aos triângulos formados pela Triangulação de Delaunay.

1.3. Definição do Problema

Para a representação de uma superfície real no computador é indispensável a elaboração e criação de um modelo digital, que pode estar representado por equações analíticas ou por um conjunto de pontos, de modo a transmitir ao usuário as características espaciais do terreno.

A criação de um modelo numérico de terreno corresponde a uma nova maneira de focar o problema da elaboração e implantação de projetos. A partir dos modelos pode-se calcular diretamente volumes, áreas, desenhar perfis e secções transversais, gerar imagens sombreadas ou em níveis de cinza, gerar mapas de declividade e aspecto, gerar fatiamentos nos intervalos desejados e perspectivas tridimensionais.

Dados de terreno são considerados de difícil tratamento devido à grande quantidade de informação necessária para a geração de modelos digitais. A gigantesca massa de dados necessária à representação digital dos dados de terreno é resultante da complexidade existente na superfície de um terreno natural.

Para ter uma boa representação digital do terreno é necessária uma grande quantidade de pontos amostrais, cujo trabalho é desgastante. Assim, procuram-se metodologias e técnicas alternativas que possam gerar malhas

3D baseadas em um pequeno número de pontos que não sejam distribuídos uniformemente.

O cuidado na escolha dos pontos e a quantidade de dados amostrados estão diretamente relacionados com a qualidade do produto final de uma aplicação sobre o modelo. Para aplicações onde se requer um grau de realismo maior, a quantidade de pontos amostrados, bem como o cuidado na escolha desses pontos, ou seja a qualidade dos dados, são decisivos. Quanto maior a quantidade de pontos representantes da superfície real, maior será o esforço computacional para que estes sejam armazenados, recuperados, processados, até que se alcance o produto final da aplicação.

1.4. Justificativa

Devido ao crescente avanço nas técnicas de sensoriamento remoto e construção de sistemas de modelagem, diversas aplicações têm utilizado dados de terreno. Estas aplicações englobam desde jogos para entretenimento até aplicações mais sérias, como Sistemas de Informação Geográfica (SIG), engenharia civil, planejamento e gerenciamento de recursos, Ciências da Terra, aplicações militares, simuladores de vôo, robótica, computação gráfica, etc.

Com a implementação de uma estrutura dinâmica não se terá uma quantidade limitada de pontos a serem inseridos e facilitará na realização de testes e verificação de hipóteses no que diz respeito à geometria da malha triangular.

Em trabalhos futuros, este estudo poderá ser utilizado para gerar a tetraedralização para que possa realizar a reconstrução volumétrica.

Para fazer a representação da superfície do terreno com uma quantidade menor de pontos amostrados, é necessário interpolar pontos para gerar a superfície.

Uma forma muito útil de se agrupar os pontos para a posterior interpolação é a triangulação desses pontos no plano e a triangulação mais utilizada é a de Delaunay.

1.5. Limitações da pesquisa

Esta pesquisa abordará o estudo sobre a Triangulação de Delaunay e a Interpolação Linear para a representação da superfície do terreno.

Não será estudado o cálculo de volume e área da superfície.

1.6. Obtenção das informações da superfície

A obtenção das informações da superfície real para fins de modelamento matemático de superfícies, consiste em levantar, por uma técnica de amostragem, um certo número de pontos com coordenadas espaciais (x, y, z) , sendo que este processo de amostragem não pode ser conduzido de forma casual. A escolha de ponto deve ser realizada de maneira que o conteúdo informativo dos mesmos represente o comportamento estrutural da superfície real. A correta definição dos pontos amostrados constituem a base de funcionamento dos algoritmos matemáticos utilizados na interpolação matemática de “alturas”.

Os pontos com suas coordenadas espaciais podem ser obtidos com base nas seguintes técnicas:

- Levantamento topográfico e geodésico;
- Fotogrametria;
- Digitalização vetorial de curvas de nível em mapeamentos analógicos;
- Transformação de curvas de nível digital, em formato vetorial, para pontos com coordenadas espaciais.

Cada uma destas técnicas possuem vantagens e desvantagens quando comparadas com as precisões obtidas as coordenadas, facilidades e tempo de execução dos trabalhos. Para a escolha de uma das técnicas deve ser levado em conta, basicamente o tipo de aplicação a que se destina o MDT (Modelagem de Terreno Digital).

2. Triangulação

A qualidade de uma superfície é medida pela forma dos seus triângulos, que se deseja aproximar-se da forma de um triângulo equilátero. Medidas típicas 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 esta relação nos triângulos equiláteros. Uma medida de qualidade da malha é então dada pelo elemento de pior medida (SHEWCHUK, 1999).

Em situações de um processo físico através do método de elementos finitos é necessário fazer a decomposição de um domínio geométrico. O tempo

para realizar a decomposição depende do número de triângulos e a estabilidade e a convergência do método são afetadas pela forma dos mesmos. Logo a qualidade de uma malha triangular deve levar em conta o número de triângulos e a forma deles.

Uma das medidas de qualidade mais empregada é o aspecto do triângulo, que é dado pela razão entre a maior aresta e a menor altura (SHEWCHUK, 1999).

A Triangulação de Delaunay é uma forma de se obter uma triangulação de um conjunto de pontos que procura otimizar algumas das medidas de qualidade desejáveis para uma malha como, por exemplo, o maior dos menores ângulos, o menor dos maiores circuncírculos, entre outros.

Existem vários algoritmos para a Triangulação de Delaunay: algoritmo incremental, dividir e conquistar, embrulho para presente, entre outros.

Nesse trabalho o algoritmo utilizado é o incremental: começando de um triângulo inicial, os pontos são inseridos, formando novos triângulos, e o critério do círculo é utilizado para realizar o *flipping* (troca de arestas quando necessário). No final, o *flipping* é realizado novamente para verificar se possíveis trocas precisam ser feitas. O resultado final desse processo é a Triangulação de Delaunay no plano.

A Triangulação de Delaunay é sempre associada ao seu dual, ou seja, a Tesselação de Dirichlet ou Diagrama de Voronoi (MESSIAS e BARBOSA, 1993).

A Tesselação de Dirichlet divide o plano em uma união de polígonos convexos de interiores disjuntos. A fronteira de cada polígono é um segmento de uma mediatriz definida por dois pontos. Cada polígono está associado a um único ponto. A Triangulação de Delaunay é construída ligando todos os pontos que compartilham a mesma fronteira de um polígono.

O critério utilizado na Triangulação de Delaunay é o de maximização dos ângulos mínimos de cada triângulo. Isto é equivalente a dizer que a malha final deve conter triângulos o mais próximo de equiláteros possível, evitando-se a criação de triângulos finos, ou seja, triângulos com ângulos internos muito agudos (SHEWCHUK, 1999).

Uma Triangulação é de Delaunay se o círculo que passa pelos três vértices de cada triângulo não contém, no seu interior, nenhum ponto do conjunto das amostras.

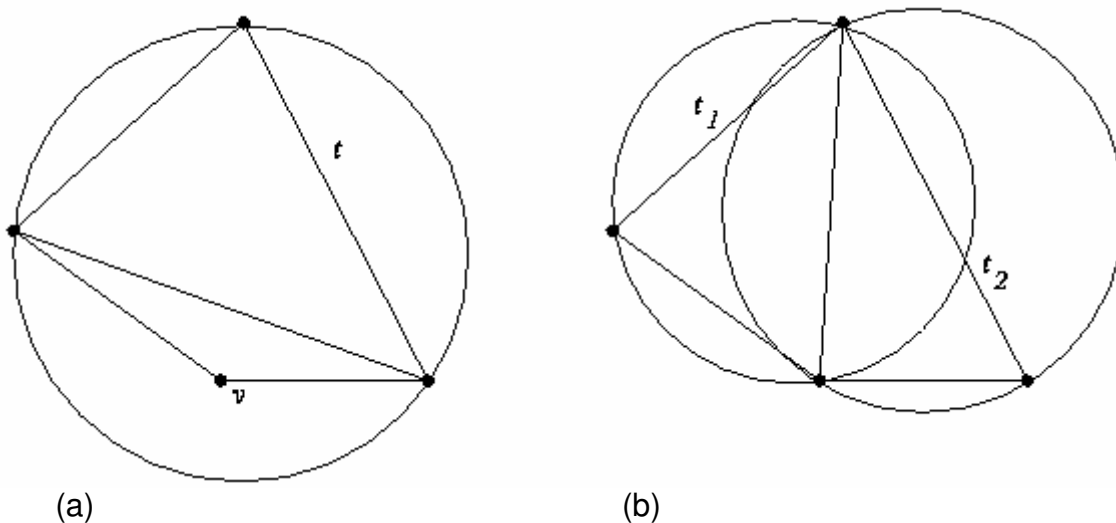


Figura 1: Critério do círculo.

Na figura 1.a), mostra que existe um ponto no interior no círculo, contrariando a regra da Triangulação de Delaunay. Na figura 1.b) a aresta é trocada, deixando os triângulos de Delaunay.

A Triangulação de Delaunay de um conjunto de pontos no plano, em geral é única, mas em alguns casos pode não ser única. Um exemplo é quando quatro pontos estão localizados no vértice de um quadrado. O único vértice da

Tesselação de Dirichlet se encontra no centro do quadrado e duas Triangulações de Delaunay são possíveis e válidas:



Figura 2: Único vértice da Tesselação de Dirichlet.

Na prática, este problema é resolvido escolhendo uma das configurações arbitrariamente. A Triangulação de Delaunay é, por definição, localmente equiangular, ou seja, ela forma triângulos os mais equiláteros possíveis.

3. Critério do círculo

Os triângulos adjacentes que têm uma aresta comum $P_1 - P_2$ e formam um quadrilátero com vértices $P_1 - P - P_2 - P_3$, como a figura abaixo, são testados para uma possível troca:

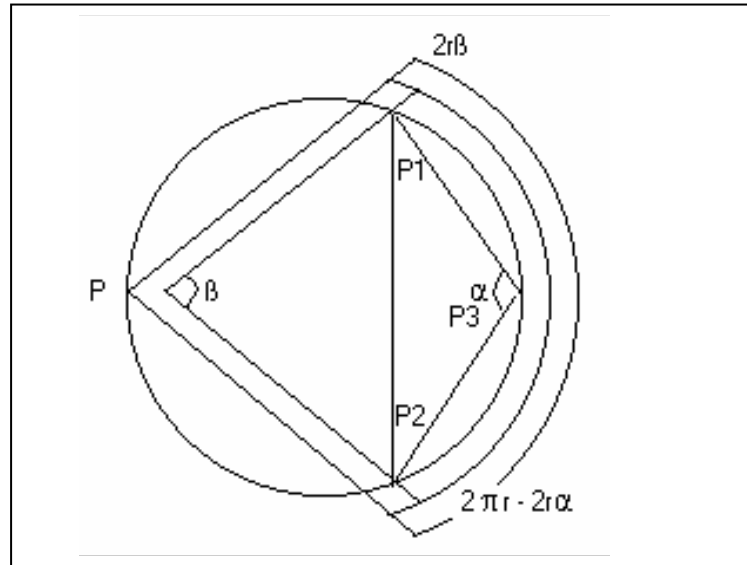


Figura 3: Critério do círculo.

A aresta P_1P_2 é trocada pela aresta PP_3 se P cai dentro da circunferência que circunscribe o triângulo $P_1P_2P_3$, ou seja, se $2\pi r - 2r\beta$, que é igual a $2r(\pi - \alpha) < 2r\beta \longrightarrow \pi < \alpha + \beta$, o ponto P pertence ao círculo, caso contrário, isto é, $\alpha + \beta < \pi$, então o ponto P cai fora do círculo e a troca não é necessária. Quando $\alpha + \beta = \pi$, então o ponto P cai sobre o círculo.

Desde que $\alpha + \beta < 2\pi$, a troca é efetuada se $\sin(\alpha + \beta) < 0$. Usando a fórmula, que é equivalente a:

$$\frac{X_{13} X_{23} + Y_{13} Y_{23}}{\sqrt{(X_{13}^2 + Y_{13}^2)(X_{23}^2 + Y_{23}^2)}} + \frac{X_{2p} X_{1p} - Y_{1p} Y_{2p}}{\sqrt{(X_{2p}^2 + Y_{2p}^2)(X_{1p}^2 + Y_{1p}^2)}} +$$

$$\frac{X_{13} Y_{23} - X_{23} Y_{13}}{\sqrt{(X_{13}^2 + Y_{13}^2)(X_{23}^2 + Y_{23}^2)}} \times \frac{X_{2p} X_{1p} - Y_{2p} Y_{1p}}{\sqrt{(X_{2p}^2 + Y_{2p}^2)(X_{1p}^2 + Y_{1p}^2)}}$$

onde:

$$X_{13} = X_1 - X_3$$

$$X_{23} = X_2 - X_3$$

$$X_{1p} = X_1 - X_p$$

$$X_{2p} = X_2 - X_p$$

$$Y_{13} = Y_1 - Y_3$$

$$Y_{23} = Y_2 - Y_3$$

$$Y_{1p} = Y_1 - Y_p$$

$$Y_{2p} = Y_2 - Y_p$$

Então, P cai dentro da circunferência se:

$$(X_{13} X_{23} + Y_{13} Y_{23})(X_{2p} X_{1p} - Y_{1p} Y_{2p}) < (X_{13} Y_{23} - X_{23} Y_{13})(X_{2p} X_{1p} - Y_{2p} Y_{1p})$$

A Triangulação de Delaunay T de um conjunto de vértices V é o grafo definido como: qualquer círculo no plano é dito vazio se ele não contém vértices de V . Se u e v são quaisquer dois vértices de V , o circuncírculo da aresta uv é qualquer círculo que passa através de u e v . Qualquer aresta tem

um número infinito de circuncírculo, mas a aresta uv está em T se e somente se existe um circuncírculo vazio de u e v . Qualquer aresta satisfazendo esta propriedade é dita Delaunay.

A figura 4 mostra uma Triangulação de Delaunay com os circuncírculos de cada triângulo.

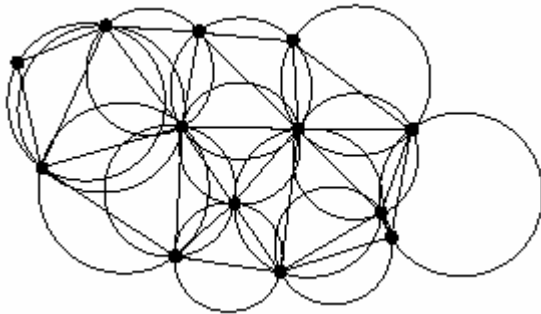


Figura 4: Triangulação de Delaunay.

Em uma triangulação, se todos os triângulos são Delaunay, então todas as arestas são Delaunay e vice-versa.

Se o triângulo T não é Delaunay, então pelo menos uma aresta não é Delaunay.

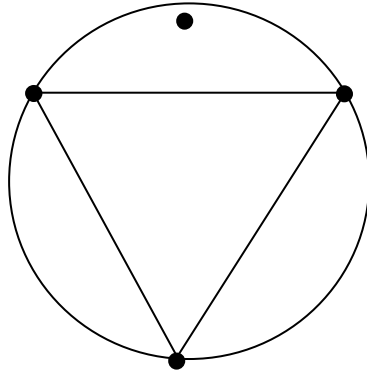
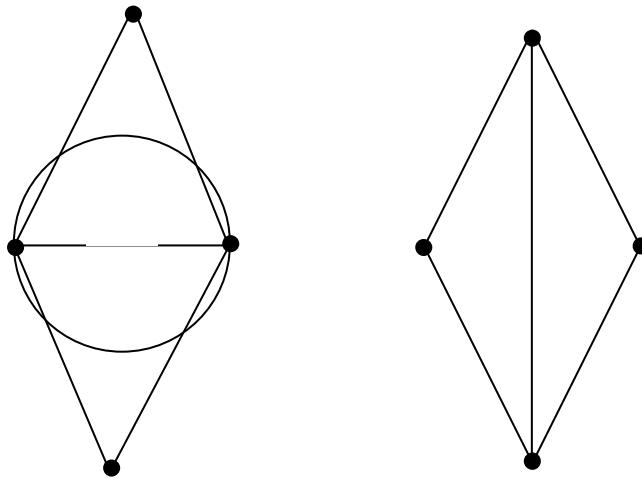


Figura 5: Triângulo que não é Delaunay.

A figura 5 mostra um triângulo que não é Delaunay.



a)

b)

Figura 6: a) Aresta localmente Delaunay, b) Aresta não localmente Delaunay.

A figura 6.a) é localmente Delaunay e a 6.b) não. Quando uma aresta não é localmente Delaunay, ela é trocada.

4. Modelo Digital de Terreno

Conforme (MITISHITA, 1997), o processo de modelagem matemática de superfícies (MDT) envolve três etapas básicas de trabalho a seguir apresentadas:

- Obtenção de informações da superfície real que possibilitem a caracterização matemática do modelo;
- Elaboração do matemático, composto por estruturas de dados e funções de interpolação que simulem o comportamento da superfície real;
- Utilização do modelo em substituição à superfície real.

A obtenção das informações da superfície real para fins de modelamento matemático de superfícies, consiste em levantar por uma técnica de amostragem um certo número de pontos com coordenadas espaciais (x, y, z) . O processo de amostragem não pode ser conduzido de forma casual. A escolha de pontos deve ser realizada de maneira que o conteúdo informativo dos mesmos represente o comportamento estrutural da superfície real. A correta definição dos pontos amostrados constituem a base de funcionamento dos algoritmos matemáticos utilizados na interpolação matemática da coordenada z de cada ponto.

Os pontos com suas coordenadas espaciais podem ser obtidos com base nas seguintes técnicas:

- Levantamentos topográficos e geodésicos;
- Fotogrametria;
- Digitalização vetorial de curvas de nível em mapeamentos analógicos;
- Transformação de curvas de nível digital, em formato vetorial, para pontos com coordenadas espaciais.

Cada uma destas técnicas possuem vantagens e desvantagens quando comparadas com as precisões obtidas as coordenadas, facilidades e tempo de execução dos trabalhos. Para a escolha de uma das técnicas deve ser levado em conta, basicamente o tipo de aplicação a que se destina o MDT.

Modelos de terreno podem ser compreendidos como uma representação digital de uma parte da superfície terrestre. Um terreno pode ser visto com uma superfície se assumirmos que cada ponto nele existente possui um único valor de altura. Através desta abstração, descartamos a presença de cavernas, grutas, e qualquer outra estrutura de maior complexidade que não satisfaça a restrição descrita acima.

O modelo digital de terreno (MDT) é termo genérico empregado para referir-se ao modelamento matemática de superfícies. Pode-se definir modelo digital de terreno como sendo um conjunto de pontos amostrados da superfície real, com coordenadas espaciais (x, y, z) determinadas num dado referencial e algoritmos que possibilitem construir um modelo matemático que reproduza da melhor maneira possível o comportamento altimétrico da superfície real (MITISHITA, 1997).

A utilização prática de MDT, até bem pouco tempo atrás, limitava-se a poucas aplicações na área de cartografia como o traçado de curvas de isovalores ou a geração de perfis altimétricos para a retificação diferencial de uma aerofoto. Contudo, com o desenvolvimento de computadores com maior

velocidade de processamento e maior capacidade de armazenamento das informações, tem-se utilizado do MDT, para a resolução de diversos problemas de engenharia que necessitam de informações do comportamento altimétrico de uma dada superfície.

5. Interpolação

O modelamento de uma superfície não consiste somente na construção de um modelo digital poliédrico. O sistema deverá possuir algoritmos de interpolação da coordenada z de cada ponto, em posições não correspondentes aos pontos amostrados. Os algoritmos devem conter certas condições de contorno, baseadas no princípio de que o comportamento de uma superfície contínua possa ser derivada do comportamento conhecido de posições próximas (MITISHITA, 1997). Geralmente, são empregados processos de interpolação que utilizam-se de uma vizinhança ilimitada de pontos, denominada de global, ou uma vizinhança limitada que é conhecida como local.

Existe um grande número de funções interpoladoras, nesse trabalho foi utilizado a Interpolação Linear.

5.1. Interpolação Linear

O esforço computacional na interpolação Linear é relativamente pequeno. Os três pontos dos vértices de cada triângulo definem um plano no espaço tridimensional. A equação do plano: $ax + by + c + d = 0$ pode ser determinada pelas coordenadas dos vértices de um triângulo de interesse. Dessa forma, para qualquer ponto a ser interpolado deve-se buscar o triângulo que o contém e, através da solução de um sistema linear, obtém-se o valor da cota do ponto. A interpolação linear garante a continuidade entre as superfícies de triângulos vizinhos, mas não garante uma suavidade na transição entre as superfícies.

Dessa forma, para cada triângulo da malha triangular, são calculadas suas coordenadas baricêntricas e, a partir delas, é calculada a função z de qualquer ponto que pertence a este triângulo.

$$f(p) = \lambda_1 f(p_1) + \lambda_2 f(p_2) + \lambda_3 f(p_3)$$

onde: $p_1, p_2, p_3 \in \mathfrak{R}^2$

$$p = (x, y) \quad p_1 = (x_1, y_1) \quad p_2 = (x_2, y_2) \quad p_3 = (x_3, y_3)$$

$$\lambda_1, \lambda_2, \lambda_3 \in \mathfrak{R}; \quad \lambda_1 + \lambda_2 + \lambda_3 = 1$$

$\lambda_i, i = 1, 2, 3$, são as coordenadas baricêntricas de p em relação a p_i .

Para determinar os λ_i é preciso resolver o seguinte sistema linear:

$$\begin{cases} \lambda_1 x_1 + \lambda_2 x_2 + \lambda_3 x_3 = x \\ \lambda_1 y_1 + \lambda_2 y_2 + \lambda_3 y_3 = y \\ \lambda_1 + \lambda_2 + \lambda_3 = 1 \end{cases}$$

Como o determinante do sistema é diferente de zero, pois representa o dobro da área do triângulo, o sistema tem solução única:

$$A = \begin{vmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix}$$

$$\lambda_1 = \frac{\begin{vmatrix} x & x_2 & x_3 \\ y & y_2 & y_3 \\ 1 & 1 & 1 \end{vmatrix}}{A}$$

$$\lambda_2 = \frac{\begin{vmatrix} x_1 & x & x_3 \\ y_1 & y & y_3 \\ 1 & 1 & 1 \end{vmatrix}}{A}$$

$$\lambda_3 = \frac{\begin{vmatrix} x_1 & x_2 & x \\ y_1 & y_2 & y \\ 1 & 1 & 1 \end{vmatrix}}{A}$$

6. Visualização da Superfície

O processo de visualização é a mudança de coordenadas do objeto (mundo) para as coordenadas do monitor do computador.

A transformação de coordenadas é baseada na proporcionalidade entre as coordenadas do objeto e do monitor:

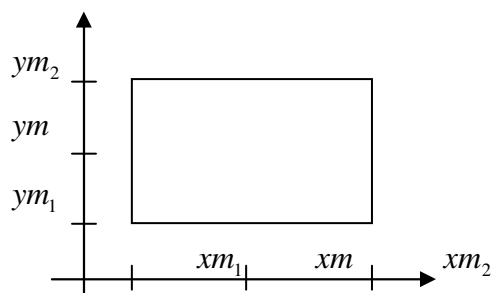


Figura 7: Sistema de coordenadas do mundo.

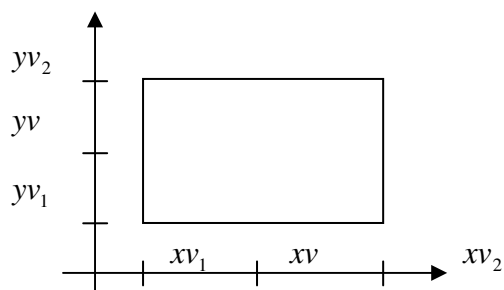


Figura 8: Sistema de coordenadas do *viewport*.

Mantendo a proporcionalidade entre os sistemas de coordenadas do mundo e do *viewport*, tem-se a seguinte relação.

$$\frac{xm_2 - xm_1}{xv_2 - xv_1} = \frac{xm_1 - xm}{xv_1 - xv}$$

$$\frac{xm_2 - xm_1}{xv_2 - xv_1} (xv_1 - xv) = xm_1 - xm$$

$$xv_1 - xv = (xm_1 - xm) \frac{(xv_2 - xv_1)}{(xm_2 - xm_1)}$$

$$xv = xv_1 - (xm_1 - xm) \text{fator_visualização_x}$$

$$\text{fator_visualização_x} = \frac{xv_2 - xv_1}{xm_2 - xm_1}$$

$$\frac{ym_2 - ym_1}{yv_2 - yv_1} = \frac{ym_1 - ym}{yv_1 - yv}$$

$$\frac{ym_2 - ym_1}{yv_2 - yv_1} (yv_1 - yv) = ym_1 - ym$$

$$yv_1 - yv = (ym_1 - ym) \frac{(yv_2 - yv_1)}{(ym_2 - ym_1)}$$

$$yv = yv_1 - (ym_1 - ym) \text{fator_visualização_y}$$

$$fator_visualização_y = \frac{yv_2 - yv_1}{ym_2 - ym_1}$$

Assim, para cada ponto (x_0, y_0) é determinado o ponto (x_v, y_v) no mundo.

7. Resultados

Essa pesquisa foi realizada com o objetivo de desenvolver uma ferramenta para modelagem digital do terreno e que permitisse ser utilizada na internet. A ferramenta foi desenvolvida em Java utilizando *Applet* que oferece o recurso para o funcionamento na web.

Foi escolhido a linguagem de programação Java por oferecer uma vasta opção de bibliotecas que satisfizeram todas as necessidades para o desenvolvimento da ferramenta.

Abaixo são mostradas algumas telas e funcionalidades da ferramenta.



Figura 9: Tela Inicial do Software.

Acima é mostrado a tela principal da ferramenta, na parte branca é a área reservada para a inserção dos pontos para a representação da superfície.

O botão “LIMPAR” que é para limpar a área branca e iniciar uma nova inserção dos pontos da superfície.

O botão “SUPERFÍCIE” é para exibir a representação digital da superfície após todos os pontos terem sido inseridos e também a malha.

A representação da malha é feita inserindo a quantidade de pontos que deseja que seja exibido no eixo x e no eixo y , após ter preenchido a quantidade dos pontos pressione o botão “EXIBIR MALHA”.

Os pontos podem ser inseridos clicando direto na área branca ou também pode ser escolhido uma configuração inicial para a representação da superfície. Para utilizar esse recurso é necessário a escolha de dois pontos e a quantidade de pontos à serem exibidos no eixo x e no eixo y . O espaço reservado para P1X e P1Y é referente as coordenadas x e y do ponto a e P2X e P2Y é referente as coordenadas x e y do ponto b , QTDX é a quantidade de pontos à serem exibidos no eixo x e QTDY é a quantidade de pontos à serem exibidos no eixo y . Após o preenchimento dos dois pontos e da quantidade de pontos do eixo x e y , pressione o botão “EXIBIR INICIALIZAÇÃO”.

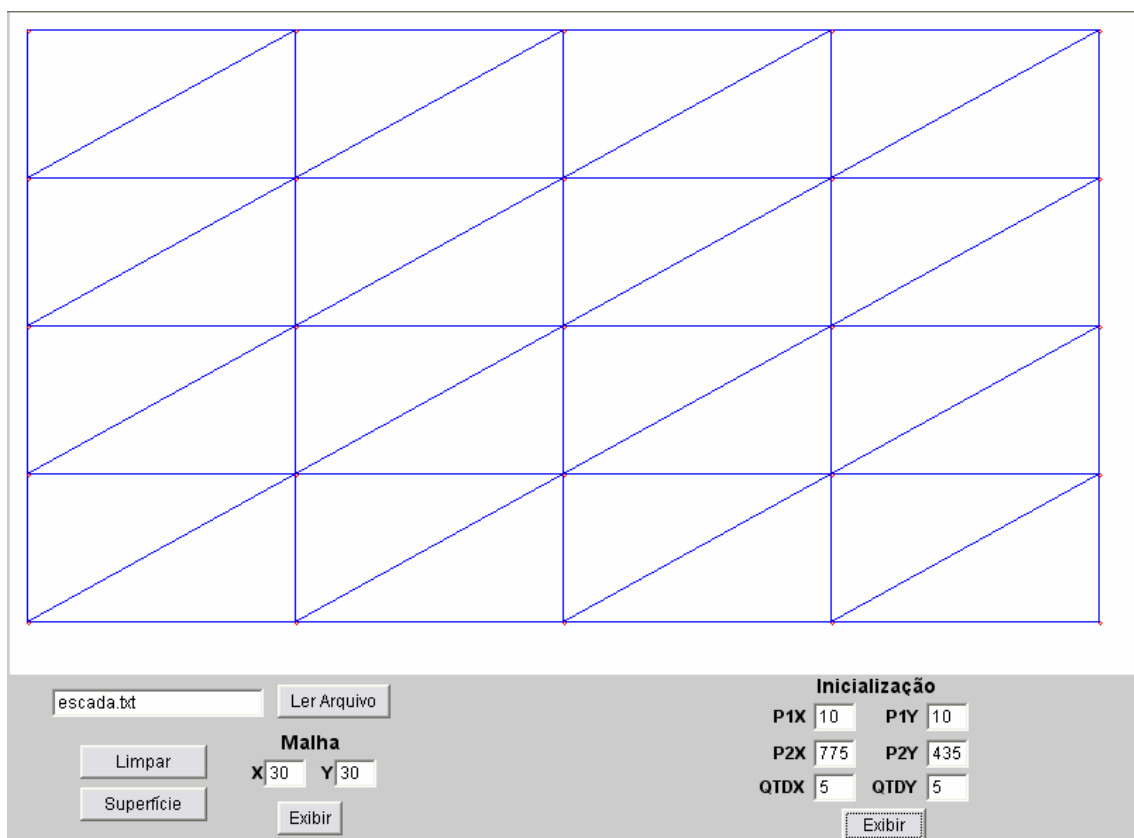


Figura 10: Triangulação de Delaunay.

A figura acima mostra os pontos inseridos pela opção de inicialização da superfície e também a Triangulação de Delaunay. Foi escolhido como pontos iniciais:

Ponto	x	y
a	10	10
b	775	435

E também foi escolhido para ser exibido 5 pontos no eixo x e 5 pontos no eixo y .

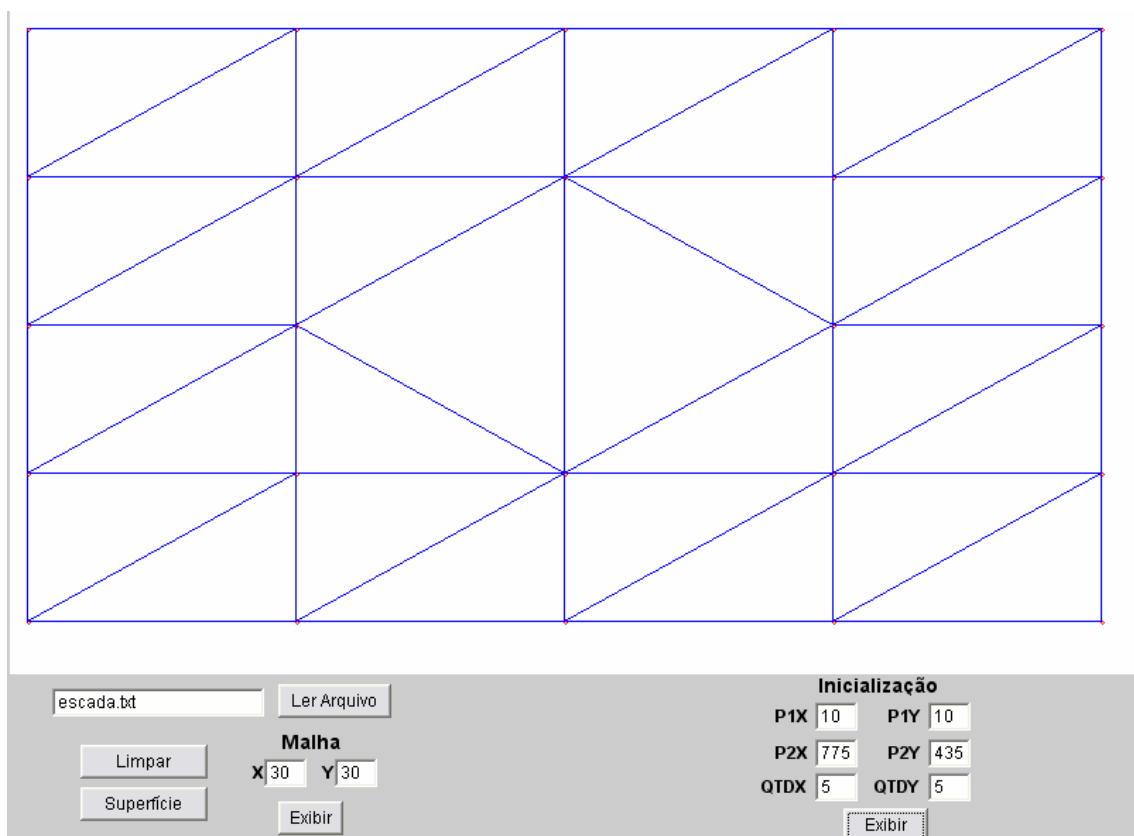


Figura 11: Triangulação de Delaunay com a remoção de um ponto.

A figura acima mostra a Triangulação de Delaunay após ter sido removido um ponto da superfície.

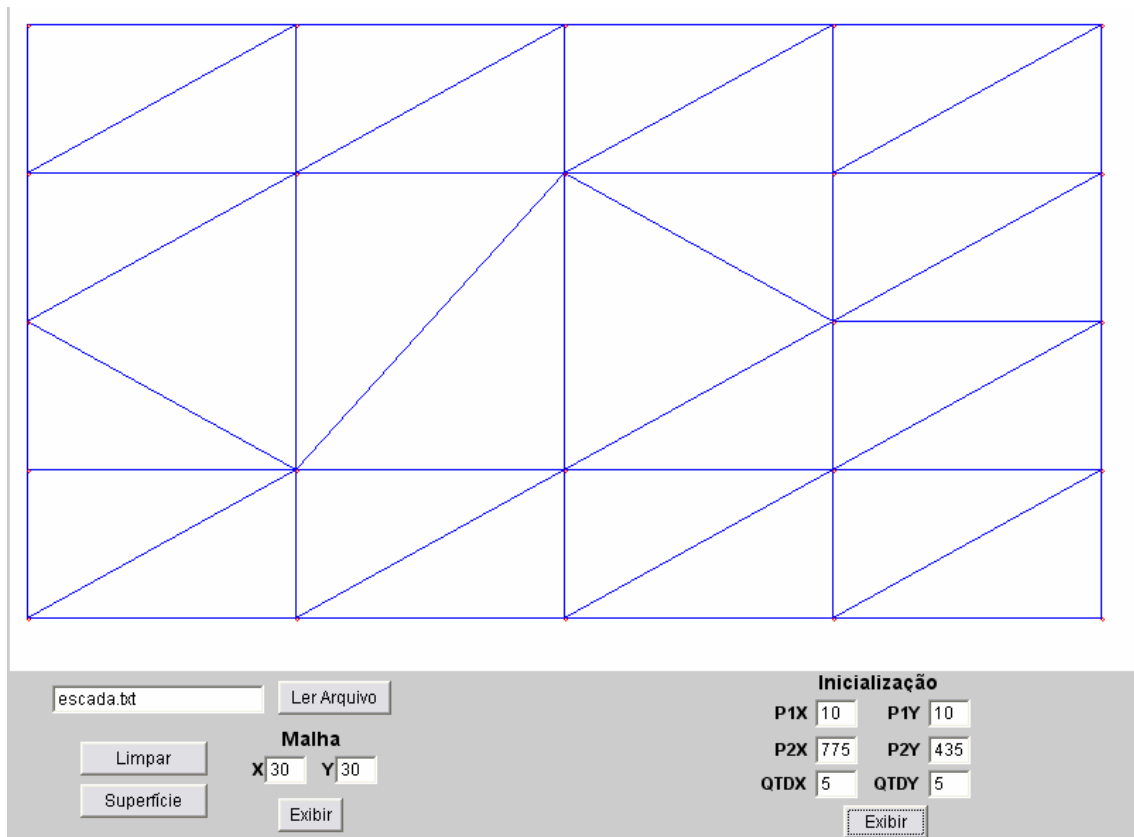


Figura 12: Triangulação de Delaunay com a remoção de dois pontos.

A figura acima mostra a Triangulação de Delaunay após ter sido removido dois pontos da superfície.

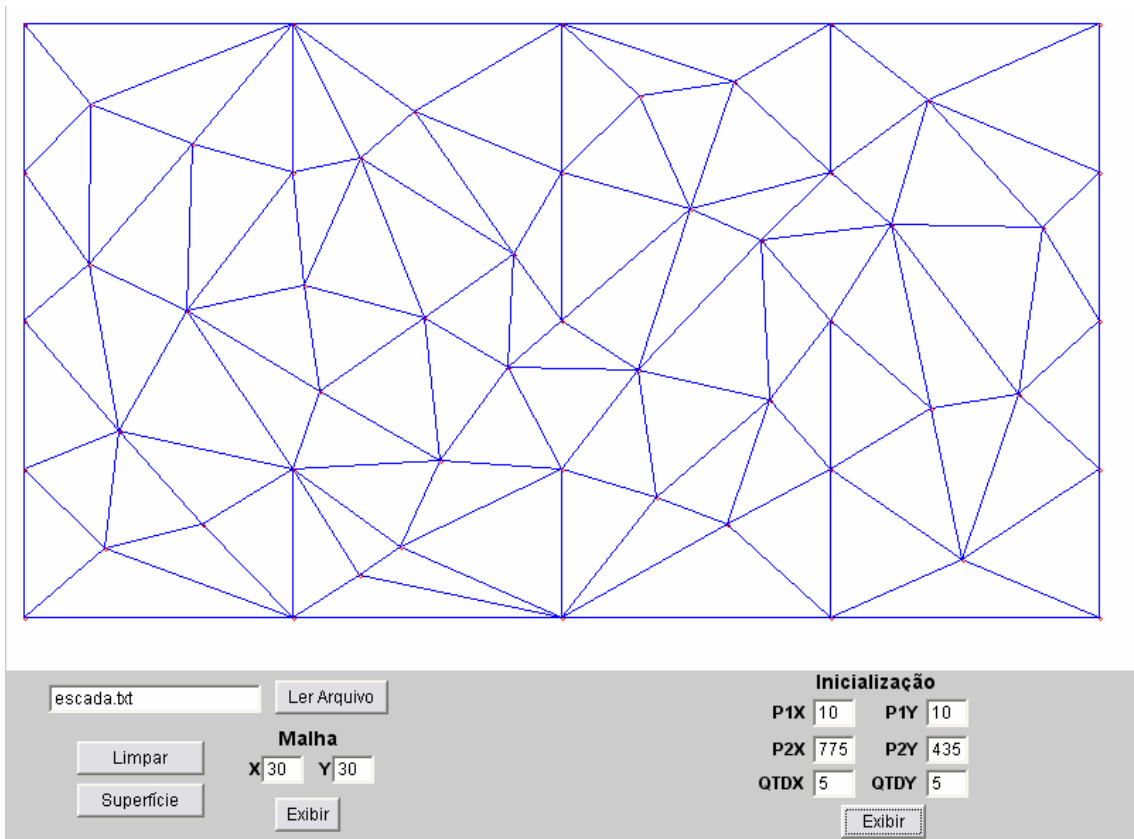


Figura 13: Triangulação de Delaunay com a inserção de vários pontos aleatórios.

A figura acima mostra a Triangulação de Delaunay após a inserção de vários pontos pelo mouse.

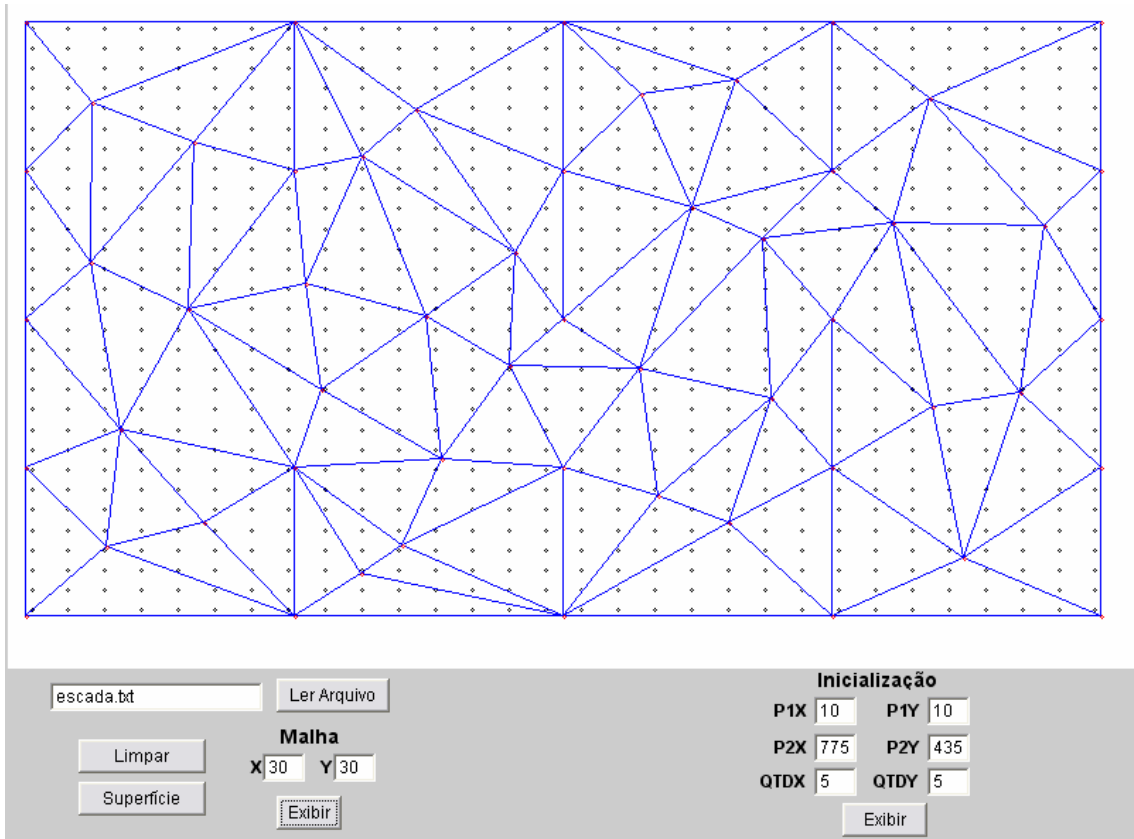
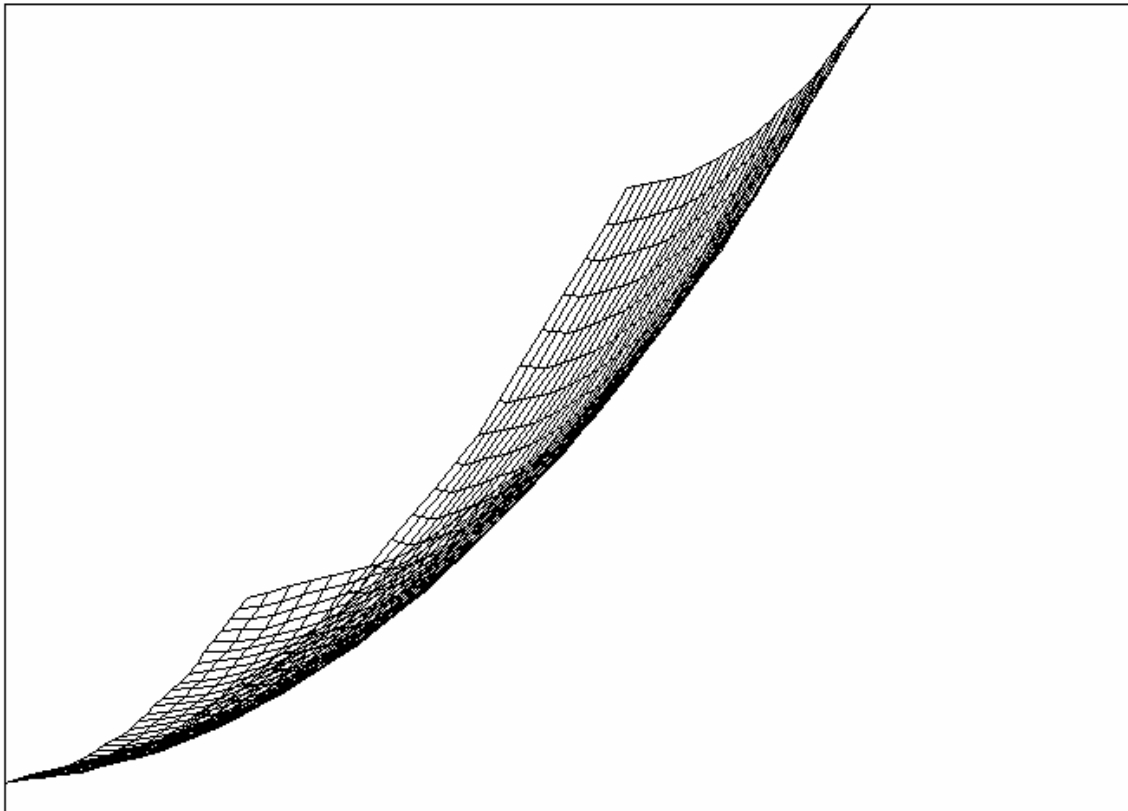


Figura 14: Malha Regular 30x30.

A figura acima mostra a Triangulação de Delaunay e a malha com 30 pontos no eixo x e 30 pontos no eixo y .



Modelagem Digital da Superfície

Figura 15: Superfície gera pela Interpolação Linear.

A figura acima mostra a superfície gerada após ter sido feito a interpolação linear de todos os pontos da malha. Essa superfície é representada por 30x30 pontos.

8. Considerações Finais

A implementação da Triangulação de Delaunay mostrou-se eficiente, pois, visualmente, as superfícies geradas apresentam-se fidedignas.

A rapidez na alteração ou remoção de pontos é uma contribuição substancial do presente trabalho.

Trabalhos futuros poderão fazer a comparação da triangulação gerada por esse programa com outros programas, para verificação da precisão e acurácia.

Outros testes também poderão ser feitos com a interpolação, verificando o erro cometido com uma superfície matemática ou com dados reais de um modelo digital do terreno.

A ferramenta mostrou-se confiável e de fácil utilização, pois apresenta comandos simples, uma interface intuitiva e funciona em qualquer *browser* (navegador de internet).

9. Bibliografia

AKIMA, H.. A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. ACM Trans. Math. Software, (Junho, 78).

MENEGUETTE, M.; BARBOSA, R. L.. Interpolação de dados não regularmente espaçados. Presidente Prudente, 1993. 84p. Monografia (Bacharel em Matemática) - Leituras e notas do Departamento de Matemática, Faculdade de Ciências e Tecnologia, Unesp de Presidente Prudente.

SHEWCHUK, J. R.. Lecture notes on Delaunay mesh generation. California at Berkeley, CA, 1999. 115p. Department of Electrical Engineering and Computer Science.

RIBEIRO, José Miguel Cardoso: A Novel Approach for Delaunay 3D Reconstruction. Disponível em: <http://ctp.di.fct.unl.pt/~ps/taaa/conteudos/show01.pdf> . Acesso em: 21/11/2004.

SAVIO, Alexandre Almeida Del: Geração da Triangulação de Delaunay. Disponível em: <http://www.tecgraf.puc-rio.br/~delsavio/GeometriaComputacional/triangle.htm>. Acesso em: 13/10/2004.

FREITAS, Eduardo Garcia de Freitas: Geometria Computacional. Disponível em: <http://www.ime.usp.br/~freitas/gc/>. Acesso em: 17/01/2005.

GUEDES, André Luiz Pires: Introdução à Geometria Computacional. Disponível em: <http://www.inf.ufpr.br/~andre/geom/>. Acesso em: 11/12/2004.

Apêndice

```
import java.awt.Graphics;

public class Ponto extends Object {
    public double x;
    public double y;
    public double z;

    public Ponto(double x, double y) {
        this.x = x;
        this.y = y;
        this.z = x * x + y * y;
    }

    public Ponto(double x, double y, int z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public boolean move(double x, double y) {
        boolean r = (this.x != x) || (this.y != y);
        this.x = x;
        this.y = y;
        z = x * x + y * y;
    }
}
```

```

        return r;
    }

    public void paint(Graphics g) {
        g.drawOval((int)x, (int)y, 2, 2);
    }
}

import java.awt.*;

public class Triangulo extends Object {

    public Ponto p1;
    public Ponto p2;
    public Ponto p3;
    public Ponto pC;
    public double R;

    public Triangulo(Ponto ap1, Ponto ap2, Ponto ap3) {
        p1 = ap1;
        p2 = ap2;
        p3 = ap3;
        double dx2 = p2.x - p1.x;
        double dy2 = p2.y - p1.y;
        double dr2 = dx2 * dx2 + dy2 * dy2;
        double dx3 = p3.x - p1.x;
        double dy3 = p3.y - p1.y;
        double dr3 = dx3 * dx3 + dy3 * dy3;
    }
}

```

```

    double A = 2 * (dx2 * dy3 - dx3 * dy2);
    double dx = (dr2 * dy3 - dr3 * dy2) / A;
    double dy = (dx2 * dr3 - dx3 * dr2) / A;
    pC = new Ponto(p1.x + dx, p1.y + dy);
    R = Math.sqrt(dx * dx + dy * dy);
}

public void paint(Graphics g) {
    g.drawLine((int)p1.x, (int)p1.y, (int)p2.x, (int)p2.y);
    g.drawLine((int)p2.x, (int)p2.y, (int)p3.x, (int)p3.y);
    g.drawLine((int)p3.x, (int)p3.y, (int)p1.x, (int)p1.y);
}

}

import java.awt.*;
import java.applet.Applet;
import java.util.Vector;

public class Superficie extends Applet {

    private CardLayout layout = new CardLayout();
    private Label texto = new Label("Modelagem Digital da Superfície");
    private SuperficieCanvas superficie = new SuperficieCanvas();

    public void init() {
        Panel menu = new Panel();
        menu.setLayout (new FlowLayout(FlowLayout.LEFT));

```

```
Panel painel = new Panel();
painel.setLayout(new BorderLayout());
painel.add("North", menu);
painel.add("South", texto);
painel.add("Center", superficie);
setLayout(layout);
add("Main", painel);
```

```
menu = new Panel ();
menu.setLayout (new FlowLayout (FlowLayout.LEFT));
```

```
painel = new Panel();
painel.setLayout(new BorderLayout());
painel.add("North", menu);
add("Help", painel);
```

```
layout.first(this);
resize (700, 600);
```

```
}
```

```
}
```

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Event;
```

```

import java.awt.Graphics;
import java.awt.Label;
import java.util.Vector;

public class SuperficieCanvas extends Canvas {
    private Vector superficie = new Vector();

    public SuperficieCanvas() {
        this.interpolacao();
    }

    private Ponto getMalha(int i) {
        return (Ponto)superficie.elementAt(i);
    }

    public void paint(Graphics g) {
        g.setColor(Color.white);
        g.fillRect(0, 0, 650, 500);
        g.setColor(Color.black);
        g.drawRect(0, 0, 650 - 1, 500 - 1);
        desenha(g);
    }

    private void interpolacao() {
        int m = MeuApplet.malha.size();
        double umax = -10e37;
        double vmax = -10e37;
        double umin = 10e37;
    }
}

```

```

double vmin = 10e37;

for (int i = 0; i < m; i++) {
    double x = ((Ponto)MeuApplet.malha.elementAt(i)).x + 0.71 *
((Ponto)MeuApplet.malha.elementAt(i)).y;
    double y = ((Ponto)MeuApplet.malha.elementAt(i)).z + 0.71 *
((Ponto)MeuApplet.malha.elementAt(i)).y;
    this.superficie.addElement(new Ponto(x, y));

    if (x < umin)
        umin = x;
    if (y < vmin)
        vmin = y;
    if (x > umax)
        umax = x;
    if (y > vmax)
        vmax = y;

}

double k1 = 500 / (umax - umin);
double k2 = 450 / (vmax - vmin);
double l1 = 500 - k1 * umax;
double l2 = 450 - k2 * vmax;

//Projeção
for (int i = 0; i < m; i++) {
    this.getMalha(i).x = (k1 * this.getMalha(i).x + l1);

```

```

        this.getMalha(i).y = (450-(k2 * this.getMalha(i).y + l2));
    }
}

public void desenha(Graphics g) {
    //Exibe a superficie
    int qtdPontosEixoX = MeuApplet.quantX;
    int qtdPontosEixoY = MeuApplet.quantY;
    for (int i = 0; i < qtdPontosEixoX; i++) {
        for (int j = 0; j < qtdPontosEixoY - 1; j++) {
            g.drawLine((int)this.getMalha(i*qtdPontosEixoY+j).x,
(int)this.getMalha(i*qtdPontosEixoY+j).y,
(int)this.getMalha(i*qtdPontosEixoY+(j+1)).x,
(int)this.getMalha(i*qtdPontosEixoY+(j+1)).y);
            g.drawLine((int)this.getMalha(j*qtdPontosEixoX+i).x,
(int)this.getMalha(j*qtdPontosEixoX+i).y,
(int)this.getMalha((j+1)*qtdPontosEixoX+i).x,
(int)this.getMalha((j+1)*qtdPontosEixoX+i).y);
        }
    }
}
}

```

```
import java.util.Vector;
```

```

public class Interpolacao {

    private final double ZERO = 0.000001;
    private Vector malha;
    private Vector triangulos;

    public Interpolacao(Vector malha, Vector triangulos) {
        this.malha = malha;
        this.triangulos = triangulos;
    }

    public double determinante(double x1, double x2, double x3, double y1,
double y2, double y3) {
        double d = (x1 * y2) + (x2 * y3) + (x3 * y1);
        d = d - (x3 * y2) - (x1 * y3) - (x2 * y1);
        if (Math.abs(d) > 10e-6)
            return d;
        else return 0;
    }

    public Vector interpolacaoBaricentrica() {
        int t = triangulos.size();
        int m = malha.size();
        if (t > 0 && m > 0) {
            for (int a = 0; a < m; a++) {
                double x = ((Ponto)malha.elementAt(a)).x;
                double y = ((Ponto)malha.elementAt(a)).y;
                for (int i = 0; i < t; i++) {

```

```

Triangulo triangulo = (Triangulo)triangulos.elementAt(i);
double pix = triangulo.p1.x;
double piy = triangulo.p1.y;
double pjx = triangulo.p2.x;
double pjy = triangulo.p2.y;
double pkx = triangulo.p3.x;
double pky = triangulo.p3.y;

double xn = (((pix-x)*(pjy-y))-((piy-y)*(pjx-x)));
double yn = (((pjx-x)*(pky-y))-((pjy-y)*(pkx-x)));
double zn = (((pkx-x)*(piy-y))-((pky-y)*(pix-x)));
if (xn < ZERO && yn < ZERO && zn < ZERO) {
    double area = this.determinante(pix,pjx,pkx,piy,pjy,pky);
    double la = this.determinante(x,pjx,pkx,y,pjy,pky) / area;
    double lb = this.determinante(pix,x,pkx,piy,y,pky) / area;
    double lc = this.determinante(pix,pjx,x,piy,pjy,y) / area;
    ((Ponto)malha.elementAt(a)).z = Math.abs(la * triangulo.p1.z) +
(lb * triangulo.p2.z) + (lc * triangulo.p3.z);
    continue;
}
}
}
}
return malha;
}
}
}

```

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;
import java.net.URL;
import java.util.Hashtable;
import java.util.Vector;

public class MeuApplet extends Applet {

    public void init() {
        initComponents();
    }

    private void initComponents() {
        panel1 = new java.awt.Panel();
        button4 = new java.awt.Button();
        button1 = new java.awt.Button();
        button2 = new java.awt.Button();
        button3 = new java.awt.Button();
        label1 = new java.awt.Label();
        label2 = new java.awt.Label();
        label3 = new java.awt.Label();
        label9 = new java.awt.Label();
        label7 = new java.awt.Label();
        label5 = new java.awt.Label();
        label4 = new java.awt.Label();
        label6 = new java.awt.Label();
    }
}
```

```
label8 = new java.awt.Label();
label10 = new java.awt.Label();
qtdX = new java.awt.TextField();
qtdY = new java.awt.TextField();
px = new java.awt.TextField();
p2x = new java.awt.TextField();
p1x = new java.awt.TextField();
p1y = new java.awt.TextField();
p2y = new java.awt.TextField();
py = new java.awt.TextField();
button5 = new java.awt.Button();

setLayout(null);

setBackground(new java.awt.Color(204, 204, 204));
addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        formMouseClicked(evt);
    }
});

panel1.setLayout(null);

button4.setLabel("EXIBIR");
button4.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(java.awt.event.MouseEvent evt) {
        button4MouseClicked(evt);
    }
}
```

```
});
```

```
panel1.add(button4);  
button4.setBounds(270, 120, 54, 24);
```

```
button1.setLabel("LIMPAR");  
button1.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {  
        button1MouseClicked(evt);  
    }  
});
```

```
panel1.add(button1);  
button1.setBounds(10, 60, 90, 24);
```

```
button2.setLabel("SUPERFICIE");  
button2.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {  
        button2MouseClicked(evt);  
    }  
});
```

```
panel1.add(button2);  
button2.setBounds(10, 90, 90, 24);
```

```
button3.setLabel("LER ARQ");  
button3.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {
```

```
        button3MouseClicked(evt);
    }
});

panel1.add(button3);
button3.setBounds(10, 120, 90, 24);

label1.setFont(new java.awt.Font("Dialog", 1, 14));
label1.setForeground(new java.awt.Color(0, 0, 0));
label1.setText("MALHA");
panel1.add(label1);
label1.setBounds(190, 100, 50, 15);

label2.setFont(new java.awt.Font("Dialog", 1, 12));
label2.setForeground(new java.awt.Color(0, 0, 0));
label2.setText("X");
panel1.add(label2);
label2.setBounds(170, 120, 10, 20);

label3.setFont(new java.awt.Font("Dialog", 1, 12));
label3.setForeground(new java.awt.Color(0, 0, 0));
label3.setText("Y");
panel1.add(label3);
label3.setBounds(220, 120, 10, 20);

label9.setFont(new java.awt.Font("Dialog", 1, 12));
label9.setForeground(new java.awt.Color(0, 0, 0));
label9.setText("QTDX");
```

```
panel1.add(label9);
label9.setBounds(530, 90, 40, 20);

label7.setFont(new java.awt.Font("Dialog", 1, 12));
label7.setForeground(new java.awt.Color(0, 0, 0));
label7.setText("P2X");
panel1.add(label7);
label7.setBounds(540, 65, 30, 20);

label5.setFont(new java.awt.Font("Dialog", 1, 12));
label5.setForeground(new java.awt.Color(0, 0, 0));
label5.setText("P1X");
panel1.add(label5);
label5.setBounds(540, 40, 30, 20);

label4.setFont(new java.awt.Font("Dialog", 1, 14));
label4.setForeground(new java.awt.Color(0, 0, 0));
label4.setText("INICIALIZA\u00c7\u00c3O");
panel1.add(label4);
label4.setBounds(570, 20, 110, 15);

label6.setFont(new java.awt.Font("Dialog", 1, 12));
label6.setForeground(new java.awt.Color(0, 0, 0));
label6.setText("P1Y");
panel1.add(label6);
label6.setBounds(620, 40, 30, 20);

label8.setFont(new java.awt.Font("Dialog", 1, 12));
```

```
label8.setForeground(new java.awt.Color(0, 0, 0));  
label8.setText("P2Y");  
panel1.add(label8);  
label8.setBounds(620, 65, 30, 20);
```

```
label10.setFont(new java.awt.Font("Dialog", 1, 12));  
label10.setForeground(new java.awt.Color(0, 0, 0));  
label10.setText("QTDY");  
panel1.add(label10);  
label10.setBounds(610, 90, 35, 20);
```

```
qtdX.setText("30");  
panel1.add(qtdX);  
qtdX.setBounds(180, 120, 30, 20);
```

```
qtdY.setText("30");  
panel1.add(qtdY);  
qtdY.setBounds(230, 120, 30, 20);
```

```
px.setText("5");  
panel1.add(px);  
px.setBounds(570, 90, 30, 20);
```

```
p2x.setText("775");  
panel1.add(p2x);  
p2x.setBounds(570, 65, 30, 20);
```

```
p1x.setText("10");
```

```
panel1.add(p1x);  
p1x.setBounds(570, 40, 30, 20);
```

```
p1y.setText("10");  
panel1.add(p1y);  
p1y.setBounds(650, 40, 30, 20);
```

```
p2y.setText("435");  
panel1.add(p2y);  
p2y.setBounds(650, 65, 30, 20);
```

```
py.setText("5");  
panel1.add(py);  
py.setBounds(650, 90, 30, 20);
```

```
button5.setLabel("EXIBIR");  
button5.addMouseListener(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {  
        button5MouseClicked(evt);  
    }  
});
```

```
panel1.add(button5);  
button5.setBounds(590, 115, 60, 24);
```

```
add(panel1);  
panel1.setBounds(0, 450, 800, 150);
```

```

}

private void button3MouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
}

private void formMouseClicked(java.awt.event.MouseEvent evt) {
    // TODO add your handling code here:
    int n = pontos.size();
    int x = evt.getX();
    int y = evt.getY();
    //verifica a distância entre o novo ponto e todos os pontos da lista
    for (int i = 0; i < n; i++)
        if (Math.abs(x - (getPonto(i).x)) < 4 && Math.abs(y - (getPonto(i).y)) < 4)
    {
        break;
    }
    if (x < menorx) menorx = x;
    else if (x > maiorx) maiorx = x;
    if (y < menory) menory = y;
    else if (y > maiory) maiory = y;
    pontos.addElement(new Ponto(x, y));
    triangulos.removeAllElements();
    paint(getGraphics());
}

private void button2MouseClicked(java.awt.event.MouseEvent evt) {

```

```

// TODO add your handling code here:
try {
    Interpolacao i = new Interpolacao(malha, triangulos);
    malha = i.interpolacaoBaricentrica();
    getAppletContext().showDocument(new URL(getCodeBase() +
"superficie.html"), "_blank");
}
catch (Exception e) {
    javax.swing.JOptionPane.showMessageDialog(null, "Erro no
endereço.");
}
}

```

```

private void button4MouseClicked(java.awt.event.MouseEvent evt) {
    if (!qtdX.getText().equals("") && !qtdY.getText().equals("")) {
        int x = Integer.parseInt(qtdX.getText());
        int y = Integer.parseInt(qtdY.getText());
        quantX = x;
        quantY = y;
        //remove todos os elementos da malha, se existir
        malha.removeAllElements();
        double distx = (((maiorx - 4) - (menorx + 4)) / (x - 1));
        double disty = (((maiory - 4) - (menory + 4)) / (y - 1));
        //adciona pontos na malha
        double i = menorx + 4;
        while (i < maiorx) {
            double j = menory + 4;
            while (j < maiory) {

```

```

        malha.addElement(new Ponto(i,j,0));
        j+=disty;
    }
    i+=distx;
}
paint(getGraphics());
}
}

```

```

private void button1MouseClicked(java.awt.event.MouseEvent evt) {
    pontos.removeAllElements();
    triangulos.removeAllElements();
    malha.removeAllElements();
    paint(getGraphics());
    menorx = 99999;
    maiorx = 0;
    menory = 99999;
    maiory = 0;
}

```

```

private void button5MouseClicked(java.awt.event.MouseEvent evt) {
    if (!p1x.getText().equals("") && !p2x.getText().equals("") &&
        !p1y.getText().equals("") && !p2y.getText().equals("")) {
        this.button1MouseClicked(evt);
        int x1 = Integer.parseInt(p1x.getText());
        int y1 = Integer.parseInt(p1y.getText());
        int x2 = Integer.parseInt(p2x.getText());
        int y2 = Integer.parseInt(p2y.getText());
    }
}

```

```

int x = Integer.parseInt(px.getText());
int y = Integer.parseInt(py.getText());
menorx = x1;
maiorx = x2;
menory = y1;
maiory = y2;
int distx = (Math.abs(((x2-2) - (x1+2)) / (x - 1)));
int disty = (Math.abs(((y2-2) - (y1+2)) / (y - 1)));
for (int i = x1+2 ; i < x2; i+=distx) {
    for (int j = y1+2; j < y2; j+=disty) {
        pontos.addElement(new Ponto(i,j));
    }
}
paint(getGraphics());
}
}

```

```

public void paint(Graphics g) {
    int n = pontos.size();
    g.setColor(Color.CYAN);
    g.fillRect(0, 0, 800, 600);

    //verifica se existe algum triangulo na lista
    if (triangulos.size() == 0) {
        //lista a tabela
        tabela.clear();
        //varre a lista dos pontos
        for (int i = 0; i < n; i++) {

```

```

Ponto pi = getPonto(i);
double pix = pi.x;
double piy = pi.y;
double piz = pi.z;
for (int j = i + 1; j < n; j++) {
    Ponto pj = getPonto(j);
    double pjx = pj.x;
    double pjy = pj.y;
    double pjz = pj.z;
    for (int k = i + 1; k < n; k++) {
        Ponto pk = getPonto(k);
        double pkx = pk.x;
        double pky = pk.y;
        double pkz = pk.z;
        //calcula lado
        double zn = (pjx - pix) * (pky - piy) - (pkx - pix) * (pjy - piy);
        if (j == k || zn > -ZERO)
            continue;

        double D = (2*(piy*pkx + pjy*pix - pjy*pkx - piy*pjx - pky*pix +
pky*pjx));
        double P0 = ((pjy*(pix*pix) - pky*(pix*pix) - (pjy*pjy)*piy +
(pky*pky)*piy + (pjx*pjx)*pky + (piy*piy)*pjy + (pkx*pkx)*piy - (pky*pky)*pjy -
(pkx*pkx)*pjy - (pjx*pjx)*piy + (pjy*pjy)*pky - (piy*piy)*pky) / D);
        double P1 = (((pix*pix)*pkx + (piy*piy)*pkx + (pjx*pjx)*pix -
(pjx*pjx)*pkx + (pjy*pjy)*pix - (pjy*pjy)*pkx - (pix*pix)*pjx - (piy*piy)*pjx -
(pkx*pkx)*pix + (pkx*pkx)*pjx - (pky*pky)*pix + (pky*pky)*pjx) / D);

```

```

        int Distancia = (int)(Math.sqrt(Math.pow((P1 - piy),2) +
Math.pow((P0 - pix),2)));

        int m = 0;
        int posZero = -1;
        for (m = 0; m < n; m++) {
            Ponto pm = getPonto(m);
            if (m != i && m != j && m != k) {
                int resul = (int)(Math.sqrt(Math.pow((P1 - pm.y),2) +
Math.pow((P0 - pm.x),2)));
                if (resul < Distancia) {
                    posZero = -1;
                    break;
                }
                else if (resul == Distancia) {
                    posZero = m;
                }
            }
        }
        if (m == n) {
            if (posZero > -1) {
                double xn1 = (getPonto(posZero).x - pix) * (pky - piy) - (pkx
- pix) * (getPonto(posZero).y - piy);
                double xn2 = (getPonto(posZero).x - pix) * (pjy - piy) - (pjx -
pix) * (getPonto(posZero).y - piy);

                double xn3 = (getPonto(posZero).x - pkx) * (piy - pky) - (pix
- pkx) * (getPonto(posZero).y - pky);

```

```
double xn4 = (getPonto(posZero).x - pkx) * (pjy - pky) - (pjx - pkx) * (getPonto(posZero).y - pky);
```

```
double xn5 = (getPonto(posZero).x - pjx) * (pky - pjy) - (pkx - pjx) * (getPonto(posZero).y - pjy);
```

```
double xn6 = (getPonto(posZero).x - pjx) * (piy - pjy) - (pix - pjx) * (getPonto(posZero).y - pjy);
```

```
if ((xn1 > 0 && xn2 < 0)||xn1 < 0 && xn2 > 0) {  
    if (!this.existeTriangulo(pk, pi, pj, getPonto(posZero))) {  
        Triangulo novoTriangulo = new Triangulo(pi, pj, pk);  
        triangulos.addElement(novoTriangulo);  
    }  
  
}  
  
else if ((xn3 > 0 && xn4 < 0)||xn3 < 0 && xn4 > 0) {  
    if (!this.existeTriangulo(pj, pk, pi, getPonto(posZero))) {  
        Triangulo novoTriangulo = new Triangulo(pi, pj, pk);  
        triangulos.addElement(novoTriangulo);  
    }  
  
}  
  
else if ((xn5 > 0 && xn6 < 0)||xn5 < 0 && xn6 > 0) {  
    if (!this.existeTriangulo(pi, pj, pk, getPonto(posZero))) {  
        Triangulo novoTriangulo = new Triangulo(pi, pj, pk);  
        triangulos.addElement(novoTriangulo);  
    }  
  
}  
  
}
```

```

        else {
            Triangulo novoTriangulo = new Triangulo(pi, pj, pk);
            triangulos.addElement(novoTriangulo);
        }
    }
}
}
}
}
}
//desenha todos os pontos da malha
g.setColor(Color.black);
for (int i = 0; i < malha.size(); i++) {
    ((Ponto)malha.elementAt(i)).paint(g);
}
//seta a cor azul
g.setColor(Color.blue);
//desenha todos os triangulos da lista
for (int i = 0; i < triangulos.size(); i++) {
    getTriangulo(i).paint(g);
}
//seta a cor vermelha
g.setColor(Color.red);
//desenha todos os pontos da lista
for (int i = 0; i < n; i++) {
    getPonto(i).paint(g);
}
}
}

```

```

private boolean existeTriangulo(Ponto pi, Ponto pj, Ponto pk, Ponto pm) {
    int i = 0;
    for (; i < triangulos.size(); i++) {
        Triangulo t = getTriangulo(i);
        if (((pi.x == t.p1.x && pi.y == t.p1.y) || (pi.x == t.p2.x && pi.y == t.p2.y) ||
(pi.x == t.p3.x && pi.y == t.p3.y)) ||
            ((pk.x == t.p1.x && pk.y == t.p1.y) || (pk.x == t.p2.x && pk.y == t.p2.y)
|| (pk.x == t.p3.x && pk.y == t.p3.y)))
            if ((pj.x == t.p1.x && pj.y == t.p1.y) || (pj.x == t.p2.x && pj.y == t.p2.y)
|| (pj.x == t.p3.x && pj.y == t.p3.y))
                if ((pm.x == t.p1.x && pm.y == t.p1.y) || (pm.x == t.p2.x && pm.y ==
t.p2.y) || (pm.x == t.p3.x && pm.y == t.p3.y))
                    return true;
        }
        return false;
    }
}

private Ponto getPonto(int i) {
    return (Ponto) pontos.elementAt(i);
}

private Triangulo getTriangulo(int i) {
    return (Triangulo) triangulos.elementAt(i);
}

// Variables declaration - do not modify
private java.awt.Button button1;
private java.awt.Button button2;

```

```
private java.awt.Button button3;
private java.awt.Button button4;
private java.awt.Button button5;
private java.awt.Label label1;
private java.awt.Label label10;
private java.awt.Label label2;
private java.awt.Label label3;
private java.awt.Label label4;
private java.awt.Label label5;
private java.awt.Label label6;
private java.awt.Label label7;
private java.awt.Label label8;
private java.awt.Label label9;
private java.awt.TextField p1x;
private java.awt.TextField p1y;
private java.awt.TextField p2x;
private java.awt.TextField p2y;
private java.awt.Panel panel1;
private java.awt.TextField px;
private java.awt.TextField py;
private java.awt.TextField qtdX;
private java.awt.TextField qtdY;
// End of variables declaration
public static Vector malha = new Vector();
public static int quantX;
public static int quantY;
private Vector pontos = new Vector();
private Vector triangulos = new Vector();
```

```
private Vector arestas = new Vector();  
private Hashtable tabela = new Hashtable();  
private double menorx = 99999;  
private double menory = 99999;  
private double maiorx = 0;  
private double maiory = 0;  
private final double ZERO = 0.000001;  
}
```