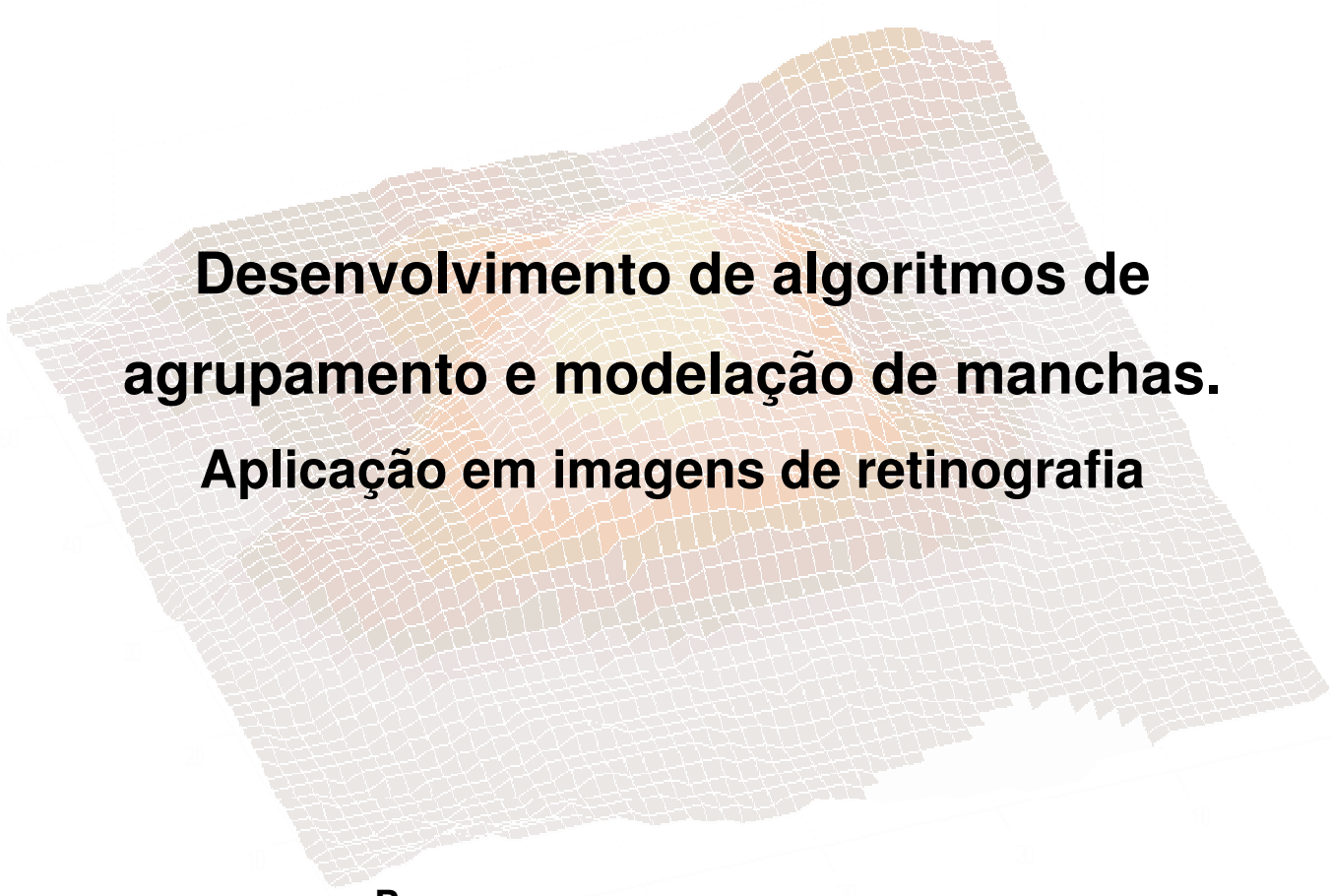
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Sistemas Digitais e Percepcionais



Desenvolvimento de algoritmos de agrupamento e modelação de manchas.

Aplicação em imagens de retinografia

Por:

Fernando Miguel Bernardo Moitinho

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Electrotécnica e Computadores

Orientador: Prof. Doutor José Manuel da Fonseca

Lisboa

2008

Agradecimentos

Sendo este trabalho a última etapa para a finalização do meu Mestrado Integrado em Engenharia Electrotécnica e de Computadores, foi por mim escolhido como sendo um que me desse particular interesse, não só profissional como também pessoal. Por conseguinte, gostaria de agradecer ao Professor José Manuel da Fonseca e ao Mestre André Damas Mora pela oportunidade de satisfazerem este meu desejo, pelo tempo despendido, pela completa disponibilidade para me elucidarem em todas as questões suscitadas e ainda pela boa disposição e convívio sempre presente.

Julgo também importante mencionar o apoio sentido por parte da minha família. Por me acompanharem nos momentos mais difíceis e de maior preocupação, o meu muito obrigado

Os meus agradecimentos ainda a todos os que me acompanharam neste difícil percurso de finalizar um Mestrado.

Resumo

A Degeneração da Mácula Relacionada com a Idade (DMRI) é uma doença dos olhos que é caracterizada pelo aparecimento de manchas de tom amarelado na retina. Estas manchas são vulgarmente chamadas de Drusas. Estas manchas são normalmente objecto de análise dos oftalmologistas de forma a verificar a eficiência dos tratamentos efectuados.

Neste trabalho é proposta uma metodologia para ajudar a comunidade de oftalmologistas a analisar as manchas de Drusas. O objectivo desta metodologia é encontrar um modelo matemático que caracterize a imagem a analisar. A partir desta técnica é possível eliminar o ruído e a análise da imagem pode ser reproduzida de forma independente do médico, permitindo ainda obter medições mais precisas do que as efectuadas manualmente.

No processo de modelação são usados diversos algoritmos tais como: algoritmo de correcção da iluminação não uniforme baseado num filtro gaussiano, algoritmo de localização das Drusas baseado em etiquetagem do caminho do maior gradiente, algoritmo de agrupamento das Drusas em imagens individuais baseado em componentes ligadas e algoritmo de optimização baseado no algoritmo de Levenberg-Marquardt.

Abstract

Age Related Macular Degeneration (ARMD) is an eye disease characterized by the appearing of yellowish spots in the retina. These spots are usually called Drusen spots.

Drusen spots are usually object of analysis by the ophthalmologists for treatment effectiveness evaluation. In this work a methodology to help ophthalmologist community in Drusen spots analysis is proposed. The objective of this methodology is to achieve a mathematical model of the original image. With this technique noise is removed, analysis can be reproduced independently of the clinician and Drusen spots measures can be done accurately. In the modelling process several image processing algorithms are used: non-uniform illumination correction, based on gaussian blur filter; Drusen location algorithm based on labelling of the maximum gradient path; grouping Drusen spots producing smaller images based in connected components labelling; Levenberg-Marquardt optimization algorithm for image modelling.

Índice

Índice de figuras

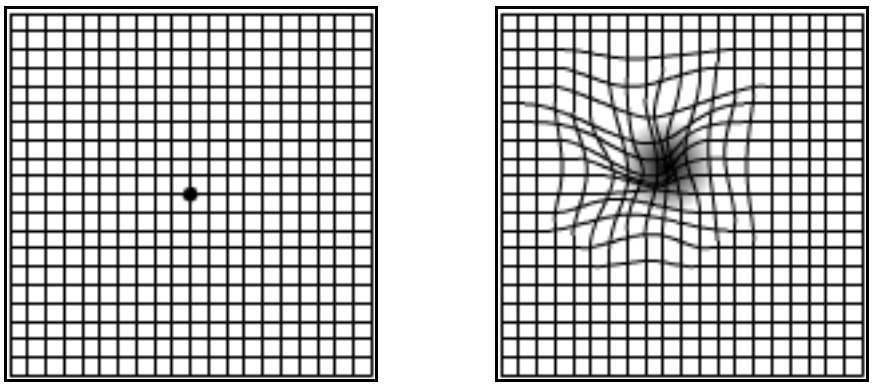
1 Introdução

No campo da oftalmologia existe uma doença denominada Degeneração Macular Relacionada com a Idade (DMRI). Esta doença é a principal causa de cegueira em indivíduos idosos [1], podendo afectar um ou ambos os olhos. Os primeiros sinais do aparecimento de DMRI podem incluir o aparecimento de um ponto preto ou apenas uma distorção na visão central. Com o tempo estes sinais tornam-se mais óbvios chegando mesmo a eliminar a visão central. Embora esta perturbação na visão central seja permanente, raramente afecta a visão periférica (Figura 1.1).

Campo Visual Normal Campo Visual afectado por DMRI

Figura 1.1 – Forma como o Campo Visual é afectado por DMRI

Existem vários testes que se podem efectuar para diagnosticar o aparecimento prévio da doença. Um destes testes consiste em olhar para um gráfico com um padrão de grelha (grelha Amsler [2]) a fim de se verificar se todas as linhas estão alinhadas e se todas as intersecções são bem visíveis. Caso contrário, poderá ser um indício de DRMI. Na Figura 1.2.a pode-se observar o exemplo de uma grelha Amsler vista por um indivíduo saudável e na Figura 1.2.b a mesma grelha como é observada por um indivíduo afectado por DMRI.



a) Individuo saudável b) Individuo afectado por DMRI

Figura 1.2 – Observação da Grelha Amsler por dois indivíduos, um saudável e outro afectado por DMRI

A DMRI desenvolve-se a partir do aparecimento de novos vasos sanguíneos anómalos numa determinada zona do olho, danificando irremediavelmente a retina e provocando consequentemente a perda da visão central. Este processo denomina-se por neovascularização coroideia (NVC).

Na Figura 1.3 pode observar-se a estrutura do olho e, mais em pormenor, a mácula.

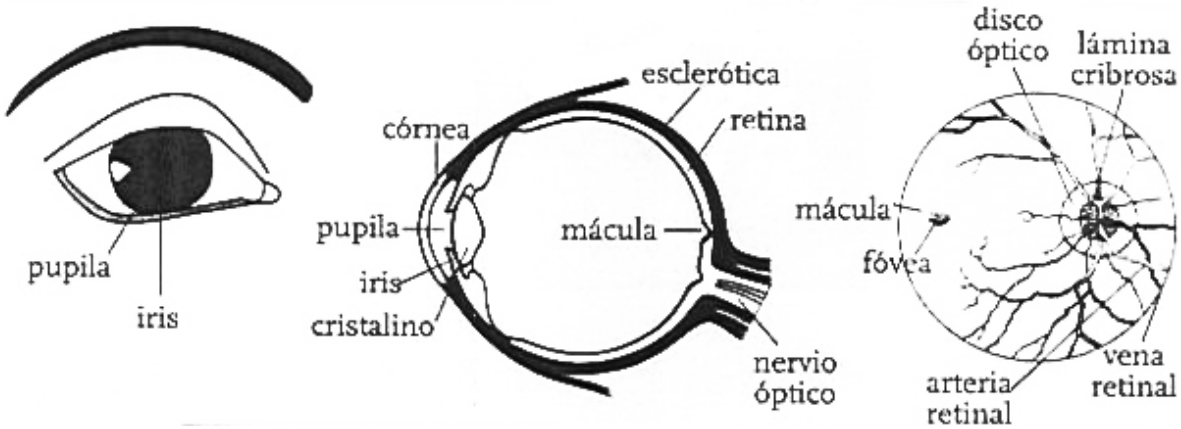
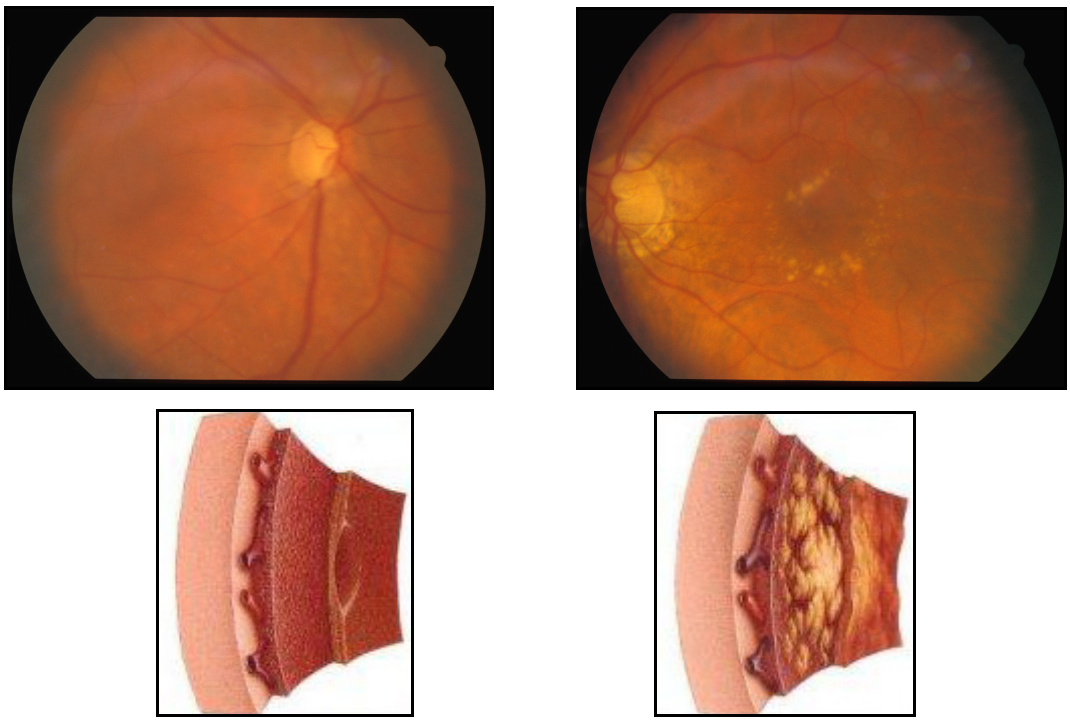


Figura 1.3 – Estrutura interna do olho [16]

Na Figura 1.4 podemos ver a diferença entre a imagem de uma retina normal e de outra com DMRI.



a) Normal b) Afectada por DMRI

Figura 1.4 – Diferença entre mácula Normal e mácula afectada por DMRI

As manchas amareladas na Figura 1.4.b) são denominadas de drusas. A detecção e a quantificação dessas manchas são essenciais para um diagnóstico precoce da anomalia, assim como para a determinação da terapia mais adequada.

Actualmente a detecção e a quantificação são feitas manualmente por inspecção visual das imagens, o que representa um trabalho monótono e de difícil reprodução. Como a análise é normalmente efectuada de forma manual pelo médico, essa análise é subjectiva e dependente do médico que a efectua. O ideal seria ter um sistema automático que realizasse esse trabalho. Assim sendo, o resultado seria preciso e independente do médico, libertando o médico desse trabalho fastidioso, o que possibilitaria fazer um diagnóstico mais exacto e possivelmente aplicar uma terapia mais adequada. Por outro lado, a construção de uma ferramenta de análise automática cria a possibilidade de comparar os resultados anteriores de modo a averiguar se a doença está a progredir ou a regredir, ajudando assim a determinar a eficácia da terapia.

É com o objectivo em criar um auxiliar para o médico oftalmologista, libertando-o do trabalho fastidioso de fazer a análise das imagens da retina para a detecção e a quantificação das drusas que surge este projecto.

O objectivo principal deste trabalho consiste em quantificar as drusas consideradas relevantes. Pretende-se, portanto, quantificar e calcular parâmetros

importantes das manchas de Drusas tais como a área e o volume de uma determinada drusa, tornando fácil e rápido o trabalho de avaliação do estado de evolução da doença.

O trabalho desenvolvido consiste numa parte de um projecto mais vasto financiado pela Fundação para a Ciência e a Tecnologia sob o número de contrato POCI/SAU-ESP/57592/2004, tem como entidades intervenientes:

- Centro de Robótica Inteligente (CRI/UNINOVA/FCT/UNL)
- Faculdade de Ciências Médicas (FCM/UNL)

2 Estado de Arte

O primeiro trabalho importante de detecção automática não supervisionada de manchas de drusas em imagens de retinografia foi publicado em 1986 por Peli and Lahav [3] tendo sido posteriormente melhorado por Sebag et al. [4] do New England Medical Center e da Tufts University School of Medicine. Este trabalho consistiu na divisão da imagem em janelas de 8x8 pixéis, calculando em cada janela um valor de binarização que posteriormente interpolado (usando um interpolação bilinear[1]) é usado na binarização da janela. Os resultados obtidos foram considerados aceitáveis, revelando no entanto, uma tendência para classificar pequenas irregularidades na imagem como Drusas (falsos positivos[2]).

Em 1991, Philips et al. [5] da Universidade de Aberdeen demonstrou que a combinação de binarizações locais e globais em imagens de funduscopia pode ser utilizada na detecção de Drusas. Mais tarde, em 1995, Kirkpatrick et al. [6] do mesmo grupo de investigação propôs a utilização de um algoritmo de *region growing* para detectar Drusas em imagens obtidas por oftalmoscopia de varrimento laser. Embora ambos produzissem resultados idênticos em imagens do mesmo paciente, os algoritmos não efectuam nenhum pós-processamento dos resultados, levando a que sejam gerados muitos falsos positivos.

Uma aproximação ao problema da detecção de Drusas usando Lógica Difusa foi apresentada em 2000 por Thdibaoui et al. [7] da Universidade de Paris. O passo inicial do algoritmo consiste em dividir os pixéis em três classes, de acordo com a sua intensidade: fundo, Drusas e ambíguos. O último passo consiste em classificar iterativamente os pixéis ambíguos numa das outras duas classes usando Lógica Difusa.

Uma segmentação de Drusas baseada numa reconstrução geodésica da imagem foi proposta em 2001 por Sbeh et al. da Universidade Paris-Dauphine [8, 9]. Este algoritmo, após aplicar um pré-processamento para o realce dos contornos, detecta máximos locais e, destes máximos e dos pixéis vizinhos extrai o fundo. Este último passo é efectuado subtraindo um valor pré-determinado ao máximo local. Os

[1] Um ponto da imagem é interpolado usando os quatro vizinhos mais próximos, Interpolando linearmente em XX e depois em YY de acordo com as proporções das dimensões em XX e YY.

[2] É denominado como falso positivo um resultado que na verdade seja negativo e o sistema dá como positivo.

resultados obtidos podem ser considerados bons, mas este algoritmo pode ser classificado como uma técnica de segmentação adaptativa e, consequentemente, tem os mesmos problemas de detecção de falsos positivos que apresentaram os trabalhos precedentes.

Com esta pequena apresentação de trabalhos relacionados pode-se concluir que outras técnicas que não de segmentação não foram ainda aplicadas a este problema. É importante fazer notar que todos estes métodos de segmentação têm a mesma tendência de produzir falsos positivos, especialmente quando a imagem tem pequenas irregularidades e que não é efectuado um pós-processamento para eliminação de manchas que não estejam relacionadas com Drusas.

3 Métodos de Obtenção de imagens da retina

Existem diversos métodos para a obtenção de imagens da retina, sendo os principais métodos a Imagem de Fundus, a Angiografia e as imagens de Varrimento Laser, os quais serão apresentados seguidamente.

3.1 Imagem de Fundus

A fotografia de Fundus é uma das formas de documentar a retina. Este método consiste em apontar uma câmara de Fundus (Figura 3.1) para a pupila sendo esta a "porta" para a entrada e saída da luz. Na Figura 3.2 pode-se observar uma fotografia da retina obtida através deste método.

Figura 3.1 – Esquema de uma câmara de Fundus [17].

As câmaras de Fundus são basicamente microscópios com uma câmara fotográfica acoplada (Figura 3.1).

Na Figura 3.3 pode observar-se o processo de entrada da luz no olho. A luz é reflectida pela retina é capturada pela câmara de Fundus.

Figura 3.2 – Imagem obtida por uma câmara de Fundus.

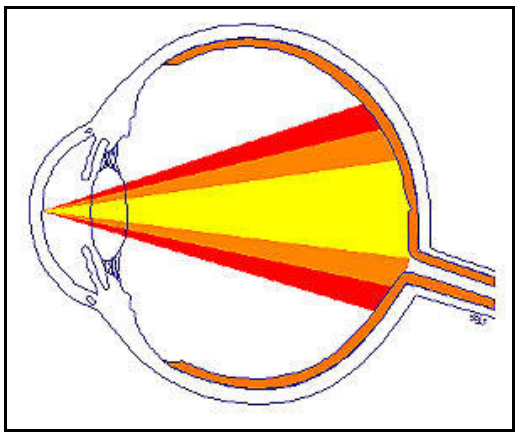


Figura 3.3 – Porção do olho captada pela câmara.

As câmaras de Fundus são geralmente diferenciadas pelo seu ângulo de captura. Um ângulo de 30º (considerado normalmente o standard) cria uma imagem duas vezes e meia maior que o original. Outras câmaras panorâmicas conseguem produzir imagens de 45º a 140º. Existem também câmaras de ângulos mais estreitos como por exemplo de 20º que permitem ver em pormenor as estruturas da retina. Na Figura 3.4 pode-se ver a diferença entre imagens produzidas por câmaras com diferentes ângulos.

30º panorâmica 20º

Figura 3.4 – Imagens captadas com câmaras de Fundus de diferentes ângulos

3.2 Angiografia

A angiografia com fluorescência é um diagnóstico de recurso que permite avaliar alguns tipos de doenças do olho. Existem dois tipos de angiografia: a angiografia com fluorescência e a angiografia com endocaína verde sendo este o tipo mais utilizado em pessoas alérgicas à fluorescência.

Em ambos os tipos de angiografia é injectado na veia do braço do paciente um elemento contrastante (circulando no organismo incluindo o olho) que evidência os vasos sanguíneos do olho.

Na angiografia com fluorescência a imagem é registada utilizando a técnica anterior, imagem de fundus, ficando assim registadas as alterações vasculares (Figura 3.5).

Na angiografia com endocaína verde é utilizada a mesma técnica mas é acoplada uma câmara de infravermelhos, pois a endocaína verde emite uma luz em infravermelhos, ficando assim registados os vasos sanguíneos e a circulação vascular.

Figura 3.5 – Imagem Captada pelo método de Angiografia

A fluorescência é um contraste amarelado, de origem vegetal, que fluoresce quando estimulada por uma luz de comprimento de onda adequado.

As principais indicações clínicas para a utilização da angiografia são a retinopatia diabética, as doenças oclusivas da retina, tais como a obstrução da veia ou da artéria retiniana e a DMRI.

Ambos os tipos de fluorescência são muito seguros pois ambos são filtrados ou pelos rins ou pelo fígado, sendo desta forma expulsos do organismo. No entanto, existem casos em que alguns pacientes apresentam náuseas e vómitos.

3.3 Varrimento Laser

A obtenção de imagens da retina por varrimento laser consiste em apontar um feixe laser de baixa potência sobre a retina, movimentá-lo em duas direcções (vertical e horizontal) e registar, numa imagem digital, a luz reflectida pela retina.

Este processo usa lasers de cores vermelha e verde podendo assim obter imagens mais reais da retina. As duas imagens captadas, uma de cor verde e outra

de cor vermelha, podem ser separadas e mostradas numa escala de cinzentos, de forma a aumentar o contraste.

Uma vez que este método usa lasers com dois comprimentos de onda diferentes, cada uma das imagens fornece informação de diferentes camadas do olho.

Na Figura 3.7 podem-se ver as duas fotografias, verde e vermelho e o resultado da sobreposição das duas.

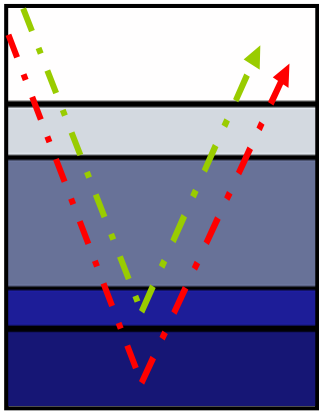


Figura 3.6 – Forma como a luz de diferentes frequências capta diferentes camadas.

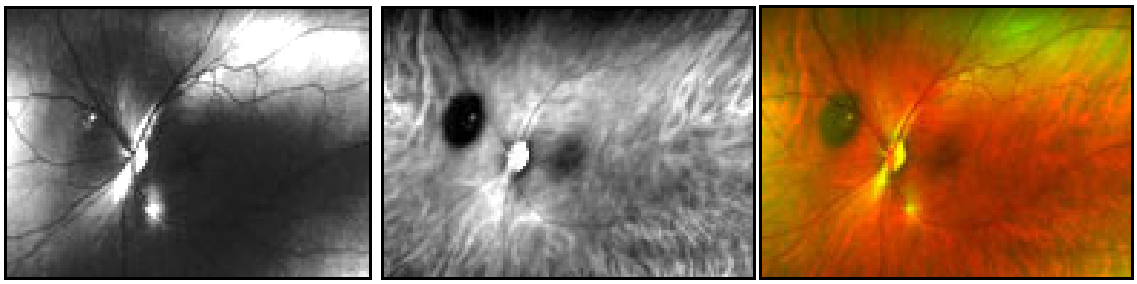


Imagem verde imagem vermelha junção das duas

Figura 3.7 – Imagens captadas por Varrimento Laser.

Usando espelhos elipsoidais é possível fazer um varrimento de 200º internos medidos a partir do centro do olho, produzindo assim imagens panorâmicas impossíveis de alcançar com os métodos convencionais (Figura 3.8). As imagens produzidas têm normalmente uma resolução de 2000x2000 pixéis.

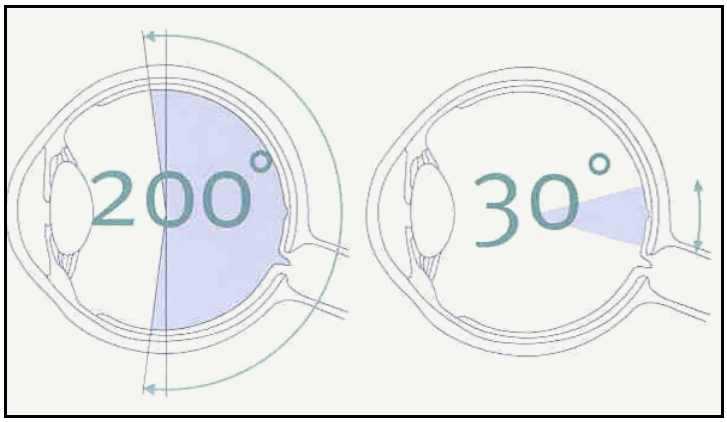


Figura 3.8 – Diferença entre o processo de Varrimento Laser e o método Fundus

Na Figura 3.9 apresentam-se alguns exemplos de imagens obtidas por varrimento laser:

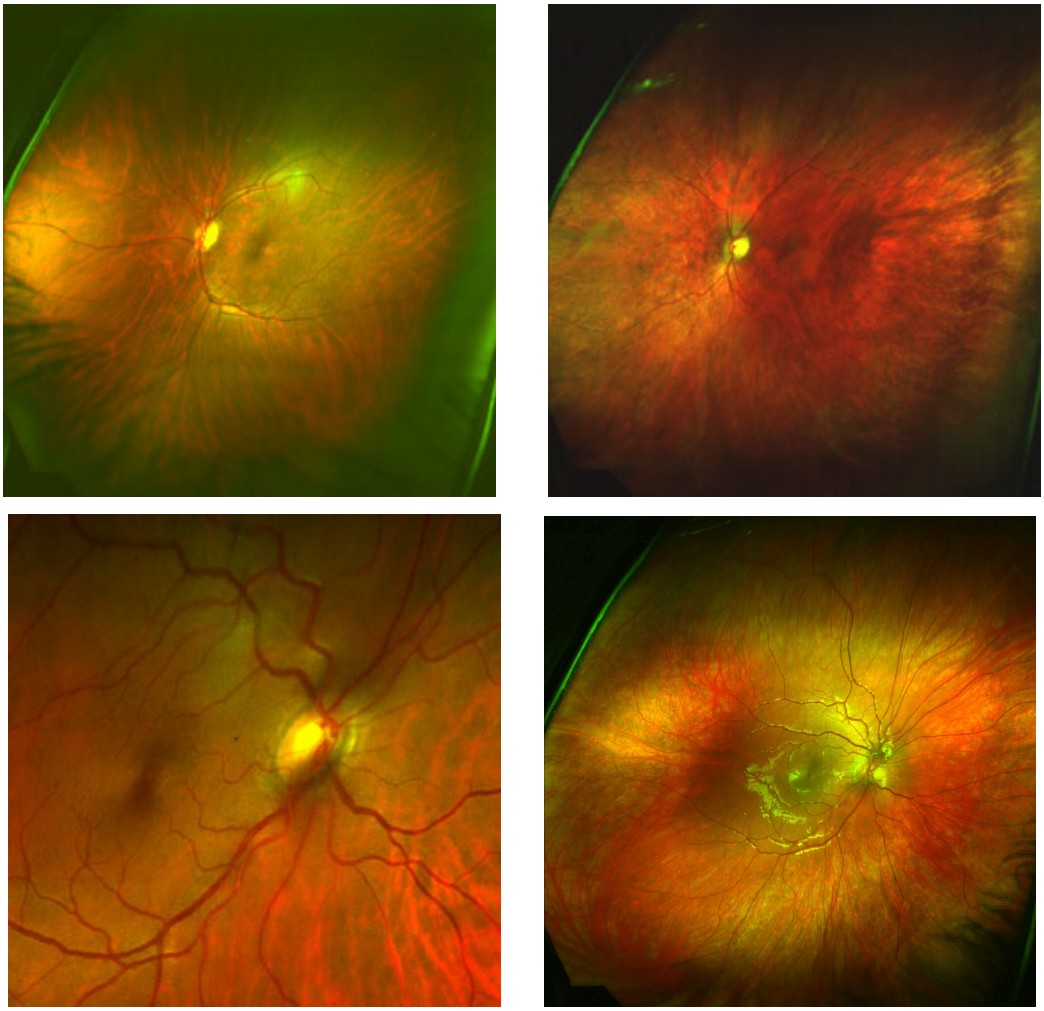


Figura 3.9 – Exemplos de Imagens de Varrimento de Laser.

Encontra-se em desenvolvimento um projecto que acrescenta mais um feixe laser, neste caso a cor azul, tornando assim possível ver mais uma camada da retina e tornando a imagem obtida mais aproximada da realidade.

Esta nova tecnologia permite obter informação de três camadas diferentes da retina permitindo identificar com mais precisão em que camada e a que profundidade se encontra determinada anomalia.

4 Trabalho desenvolvido

De modo a atingir o objectivo foram utilizados diversos algoritmos que serão explicados detalhadamente posteriormente neste capítulo. Sucintamente, a metodologia utilizada foi (Figura 4.1):

a) Pré processamento da imagem original.
b) Uniformização da iluminação.
c) A imagem resultante é dividida em pequenas imagens contendo apenas as manchas de drusas (agrupamento de Drusas).
d) Essas imagens são então individualmente processadas da seguinte forma: é executado um algoritmo para encontrar os máximos e desta forma é encontrado o centro de cada Drusa (detecção de Drusas).
e) Com o algoritmo de optimização surge a "transformação" da Drusa numa ou mais funções matemáticas, ficando assim com um modelo matemático de cada imagem (modelação de Drusas).

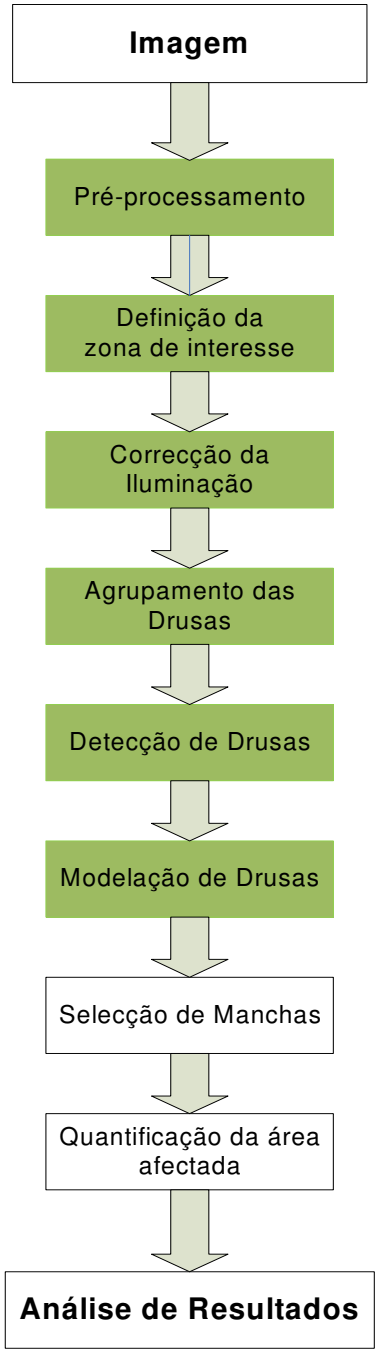


Figura 4.1 Metodologia usada para alcançar os objectivos.

Na Figura 4.1 as etapas demarcadas a verde são as que foram desenvolvidas neste trabalho, as restantes etapas fazem parte do projecto "mãe" em que este está envolvido.

A técnica de modelação de uma imagem digital tem como objectivo a reprodução da imagem através de equações matemáticas, possibilitando assim cálculos de volumes, áreas, etc..

As funções matemáticas mais utilizadas para efectuar modelações no caso das manchas de Drusas são as curvas de Gauss e de Spline[1]. Estas equações têm diversos parâmetros que se podem ajustar de forma a representarem com mais

[1] Um spline é uma curva definida matematicamente por dois ou mais pontos de controlo. Os pontos de controlo que ficam na curva são chamados de nós.

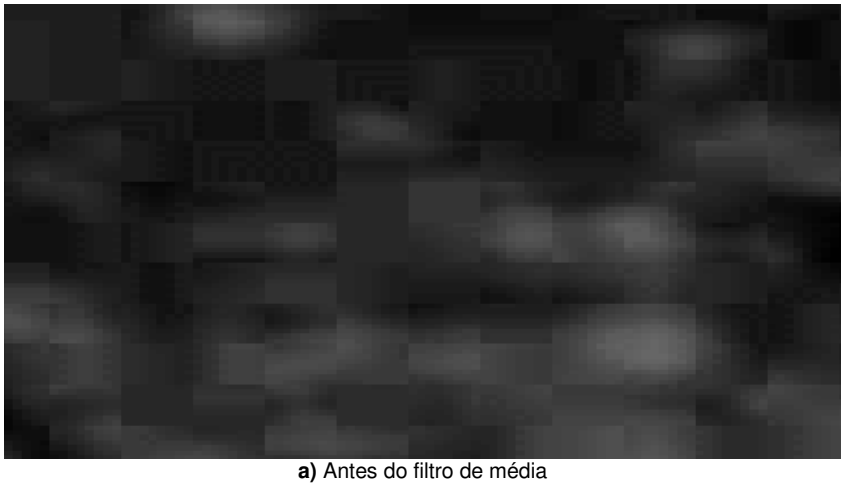
exactidão a imagem real. A modelação consiste portanto em ajustar os parâmetros automaticamente de forma a minimizar o erro entre a imagem original e a imagem modelada.

Uma imagem modelada pode ter mais do que uma função matemática de forma a obter a forma desejada.

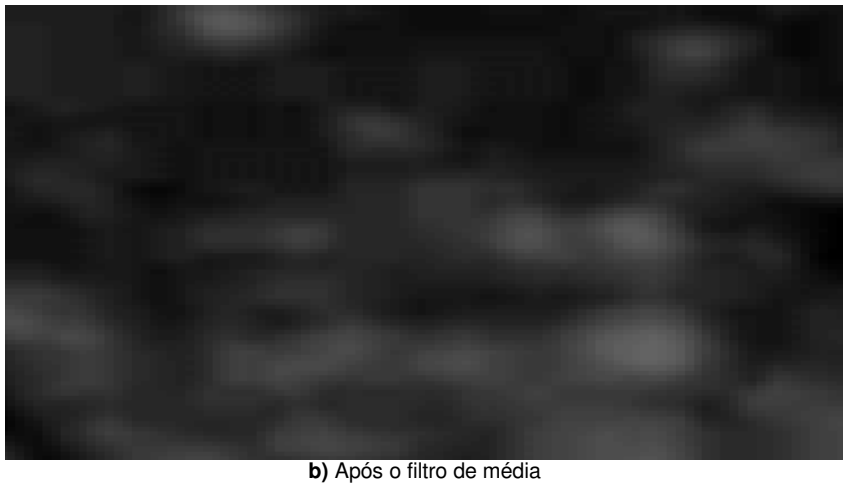
Um dos algoritmos, mais comuns, de ajuste de parâmetros (fitting) foi desenvolvido por *Levenberg e Marquardt.* Este método, que foi utilizado, será explicado mais à frente nesta secção.

4.1 Pré-Processamento

A observação de várias imagens tal como foram captadas, evidenciaram alguns defeitos que deveriam ser corrigidos antes de se poder trabalhar a imagem correctamente. Um dos defeitos encontrados é o facto de algumas imagens virem no formato JPEG, o que provoca um efeito de mosaico. Para solucionar este problema, foi aplicado um filtro de média sobre as imagens de forma a obter uma imagem mais uniforme (Figura 4.2).



a) Antes do filtro de média

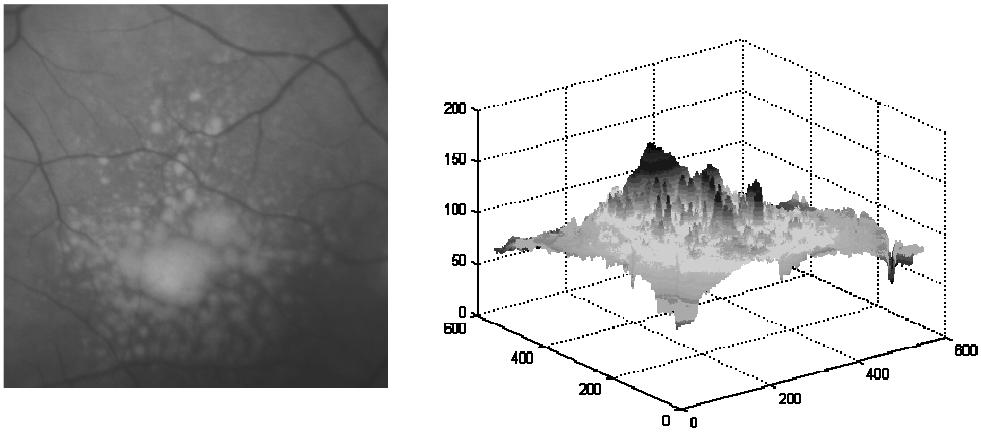


b) Após o filtro de média

Figura 4.2 - Aplicação do filtro de média.

Outro passo fundamental para o trabalho desenvolvido é a visualização das imagens num sistema de eixos XYZ, de forma a poder ser visualizado o volume das Drusas assim como outras características importantes das mesmas.

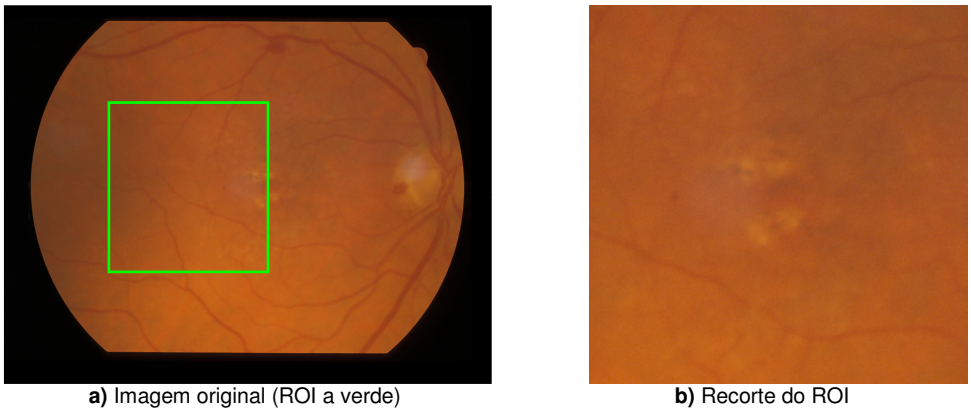
Para alcançar esse objectivo, a imagem foi colocada num sistema de coordenadas XYZ e o valor Z foi igualado à intensidade de cada pixel, resultando assim uma representação tridimensional da imagem. O resultado pode ser observado na Figura 4.3.



a) Vista 2D **b)** Vista 3D

Figura 4.3 – Representação 3D de uma imagem de retinografia

Por fim, devido à complexidade dos algoritmos usados e pelo tamanho das imagens fornecidas influenciarem proporcionalmente o tempo de processamento, apenas foram processadas as Regiões de Interesse (ROI[1]). Um exemplo do recorte efectuado pode ser observado na Figura 4.4.



a) Imagem original (ROI a verde) **b)** Recorte do ROI

Figura 4.4 – Recorte da região de interesse (ROI)

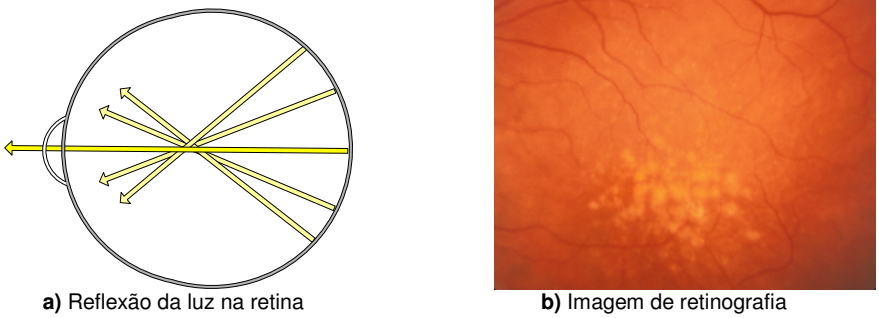
[1] Region Of Interest

Posteriormente, após o processamento da ROI, procedeu-se à junção do mesmo com a imagem original.

4.2 Correcção da Iluminação

Após a observação de várias imagens de retinografia chegou-se à conclusão que seria necessário proceder a uma uniformização e normalização da iluminação, de modo a poder efectuar a comparação entre múltiplas imagens. Para que seja possível avaliar as diferenças entre imagens de um paciente, de forma a determinar a eficácia do tratamento administrado pelo médico oftalmológico.

Como a retina não é uma superfície plana, a luz incidente não é reflectida para o mesmo ponto (Figura 4.5.a), formando assim imagens mais claras ao centro e mais escuras nos cantos, como pode ser observado na Figura 4.5.b.



a) Reflexão da luz na retina **b)** Imagem de retinografia

Figura 4.5 – Imagem exemplificativa de iluminação não uniforme

Para a resolução deste problema foram estudadas diversas abordagens, nomeadamente:

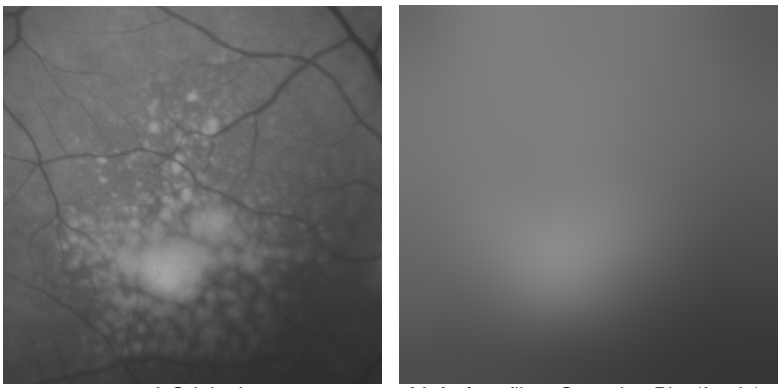
- Filtro *Gaussian Blur.*
- Espaços de cor alternativos.
- *Splines*.
- Filtros baseados em processamento na frequência.

4.2.1 Filtro *Gaussian Blur*

O filtro Gaussian Blur consiste em passar uma máscara em que os pesos atribuídos a cada pixel são calculados pela seguinte equação:

$$G(u,v) = \frac{1}{2.\pi.\sigma^2}.e^{\frac{-\left(u^2+v^2\right)}{2.\sigma^2}} \qquad \textbf{Equação 1}$$

O parâmetro u corresponde à deformação em YY e v corresponde à deformação em XX. O parâmetro σ é o desvio padrão da distribuição desejada. Quanto maior forem os parâmetros u, v e o desvio padrão (σ) menos nítida ficará a imagem. O resultado será uma imagem com um efeito de *blur* ou desfoque, que no caso específico eliminará todos os aspectos do detalhe da imagem e ficará apenas com os aspectos mais grosseiros, ou seja, a iluminação de fundo. Na Figura 4.6 pode ser visualizado o resultado da execução do algoritmo sobre uma imagem com uma iluminação de fundo não uniforme.



a) Original **b)** Após o filtro *Gaussian Blur* (fundo)

Figura 4.6 – Exemplo do filtro *Gaussian Blur*

De forma a uniformizar a iluminação é necessário remover o fundo (Figura 4.6b) à imagem original. Esta operação é efectuada dividindo a imagem original pela imagem de fundo e de seguida multiplicado por um factor de 100 de modo a que a imagem resultante fique centrada em 100 ($OriginalSemFundo = \frac{Original}{Fundo} \times 100$ Equação 2).

$$OriginalSemFundo = \frac{Original}{Fundo} \times 100 \qquad \textbf{Equação 2}$$

A imagem resultante é multiplicada por um factor de 100 apenas para facilitar alguns algoritmos implementados posteriormente. O resultado pode ser observado na Figura 4.7.

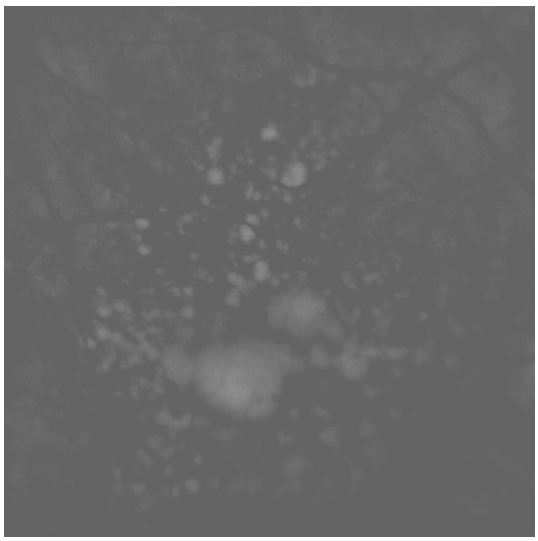


Figura 4.7 – Imagem após a correcção da iluminação pelo método *Gaussian blur*

Esta técnica apresentou resultados bastante aceitáveis não perdendo informação significa sobre as Drusas. Este foi o algoritmo escolhido para o restante trabalho desenvolvido, pois dos algoritmos testados foi o que apresentou melhores resultados.

4.2.2 Espaços de Cor

Por observação de várias imagens de retinografia tornou-se evidente que existe uma cor predominante nas mesmas, sendo assim, foi investigada a hipótese de usar essa predominância para fazer o cálculo do fundo de modo a proceder à uniformização.

O espaço de cor explorado foi o CMYK[1], o qual é calculado a partir do RGB com as seguintes equações:

$$cyan = 255 - red$$
$$magenta = 255 - green$$
$$yellow = 255 - blue$$
$$black = \min(cyan, magenta, yellow)$$

[1] CMYK é a abreviatura do sistema de cores formado por Ciano (*Cyan*), Magenta (*Magenta*), Amarelo (*Yellow*) e Preto (*blacK*).

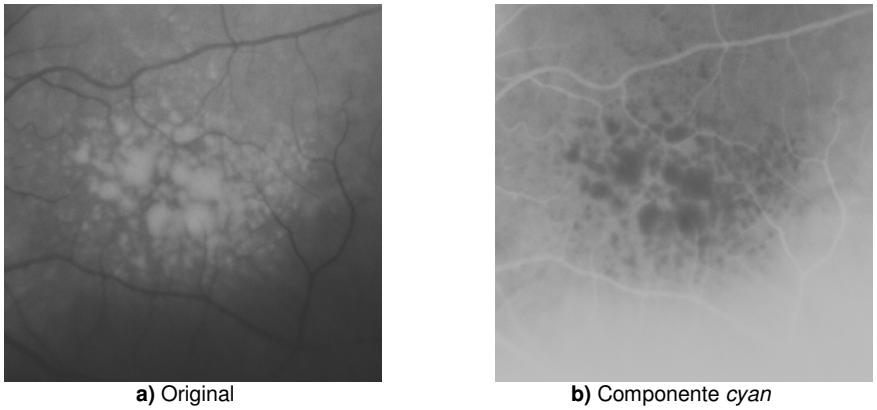
$$cyan = \frac{cyan - black}{255 - black} \times 255$$

$$magenta = \frac{magenta - black}{255 - black} \times 255$$

$$yellow = \frac{yellow - black}{255 - black} \times 255$$

As equações apresentadas são para imagens de 8 bit, ou seja, têm 255 níveis de intensidade, o que explica o número 255 nas equações.

O objectivo desta mudança de espaço é isolar a componente *cyan* do mesmo, pois é nesta componente que mais se evidencia o fundo da imagem. Pode ser observado na imagem seguinte a componente *cyan* (Figura 4.8.b) da imagem original localizada à esquerda.



a) Original **b)** Componente *cyan*

Figura 4.8 – Exemplo da aplicação da mudança do espaço de cor

Antes de proceder à remoção do fundo, é feita a seguinte operação sobre a mesma $f_b(x, y) = 255 - f_a(x, y)$. De seguida, é efectuado uma binarização com um *threshold* de 100, ou seja, os pixéis que tiverem um valor de intensidade maior que 100 ficam com o seu valor original. Caso contrário, o seu valor é substituído pelo valor 100. O resultado pode ser observado na Figura 4.9.

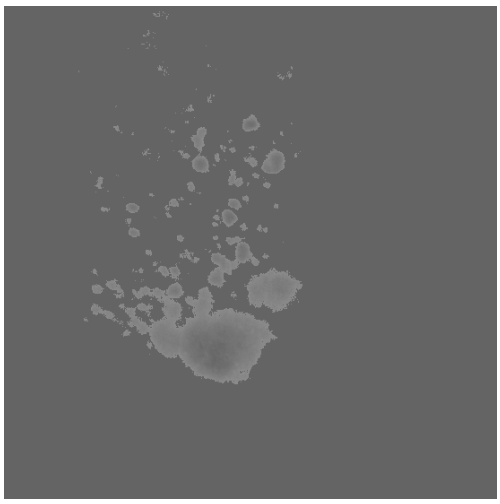


Figura 4.9 – Imagem com correcção da iluminação pelo método de espaço de cor

Esta técnica foi abandonada pois os resultados apresentados não eram satisfatórios.

4.2.3 *Smoothing Splines*

Foi testado também um algoritmo de *smoothing splines*. O objectivo da aplicação deste algoritmo é obter um efeito similar ao algoritmo *Gaussian Blur* explicado anteriormente nesta secção. O algoritmo *Smoothing Splines* utiliza dois parâmetros ajustáveis de modo a obter uma imagem com mais ou menos detalhe.

O algoritmo contém os seguintes parâmetros de ajuste:

- ***Grid:*** que representa o intervalo de amostragem e consiste numa grelha que fornece os pontos de interesse para o algoritmo tirar valores. Quanto maior for este valor menos exactidão terá ao representar a imagem (amostragem da imagem).
- ***S:*** este parâmetro representa a taxa de suavização pretendida. Quanto maior, mais suavizada ficará a imagem.

Para obter os resultados desejados foi escolhido um valor de 10000 para o parâmetro **S** e 15 para a **Grid**. Os resultados obtidos podem ser observados na Figura 4.10.

a) Original **b)** Após *Smoothing Splines* (fundo)

Figura 4.10 – Após a aplicação do algoritmo *Smoothing Splines*

O fundo foi removido de forma idêntica à utiliza no algoritmo anterior (*Gaussian Blur*), Usando a $OriginalSemFundo = \frac{Original}{Fundo} \times 100$ Equação 2 foi obtida a imagem da Figura 4.11.

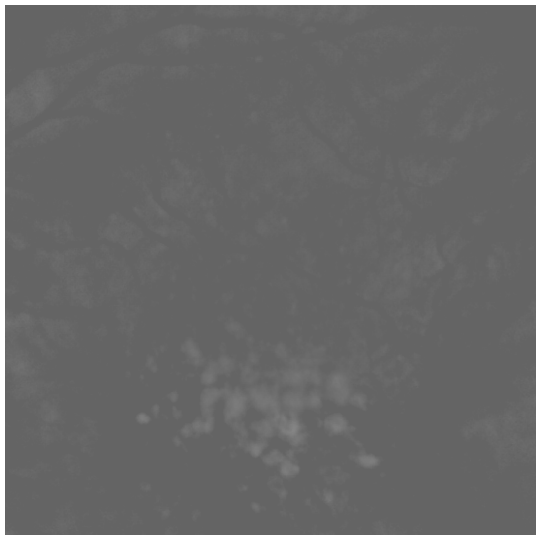


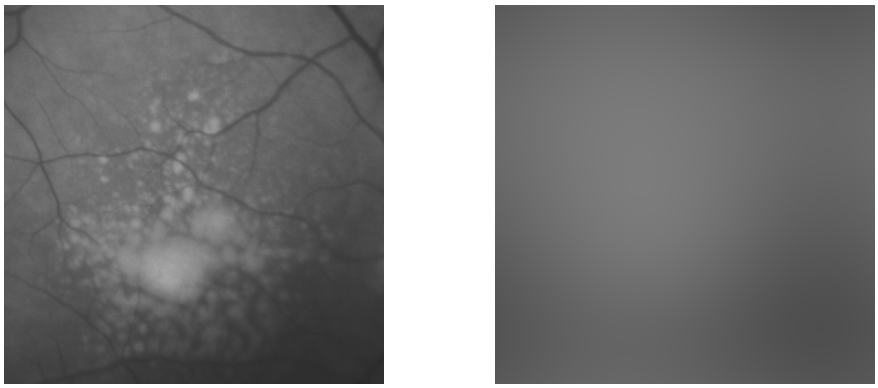
Figura 4.11 – Imagem com correcção da iluminação pelo método de *Smoothing Splines*

Esta técnica mostrou bons resultados embora se achasse que o algoritmo *Gaussian Blur* apresentou melhores resultados.

4.2.4 Processamento na Frequência

A técnica do Processamento em Frequência (PF) foi usada visando implementar um filtro passa-baixo[1]. O objectivo da utilização do filtro passa-baixo foi deixar passar as frequências baixas cortando as frequências mais altas. As frequências. Cortando as altas frequências ficamos com uma imagem apenas composta pelo fundo (Figura 4.12.b, Figura 4.13).

Recorreu-se à ferramenta MatLab para implementar um filtro passa-baixo. O algoritmo primeiro calcula a transformada de *Fourier* do sinal (imagem) e elimina as altas frequências. O resultado pode ser observado a seguir na Figura 4.12.



a) Original **b)** Após filtro passa-baixo

Figura 4.12 – Após a aplicação do filtro passa-baixo

De seguida é retirado o fundo A imagem original, dividindo a imagem original pela imagem de fundo e de seguida multiplicando o resultado por 100, de forma a centrar a imagem resultante em torno do valor 100, esta operação é representada pela $OriginalSemFundo = \frac{Original}{Fundo} \times 100$ Equação 2.

[1] Filtro passa-baixo é o nome comum dado a um circuito electrónico ou algoritmo que permite a passagem de baixas frequências e atenua (ou reduz) a amplitude das frequências altas superiores à frequência de corte. A atenuação para cada frequência varia de filtro para filtro.

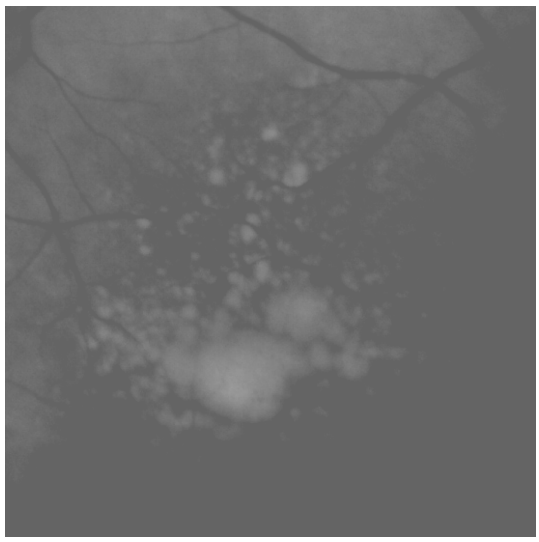


Figura 4.13 – Imagem com correcção da iluminação pelo método de PF

Este processo para correcção da iluminação encontra-se ainda em fase de investigação visto os seus resultados serem considerados aceitáveis.

4.3 Agrupamento de Drusas

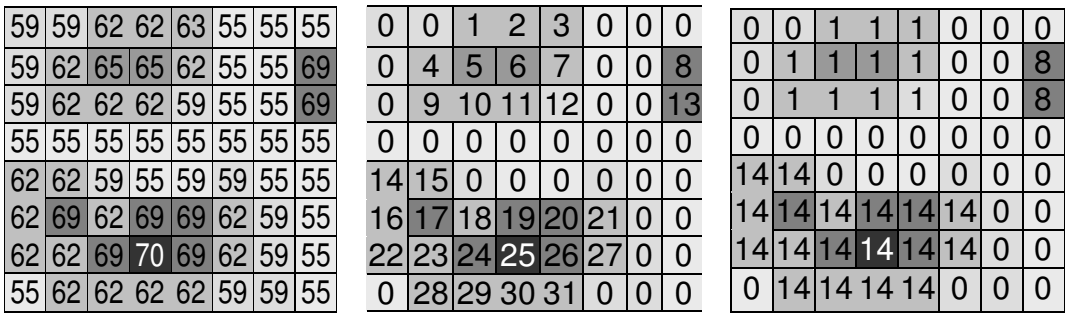
De modo a optimizar o processo de modelação das manchas de Drusas foi decidido agrupar as manchas obtendo desta forma imagens mais pequenas compostas apenas de Drusas próximas umas das outras como "arquipélagos".

Para obter o resultado pretendido foi desenvolvido um algoritmo baseado no algoritmo iterativo de componentes ligados[1] com adjacência a 8 pixéis.

A imagem resultando da uniformização da iluminação tem um valor médio do fundo. Esse valor é o *threshold* usado para determinar o que é (ou não) uma possível mancha de Drusa. Desta forma, o algoritmo de componentes ligados coloca uma etiqueta em cada pixel que satisfaça a condição:

- valor da Intensidade do pixel maior que o valor da intensidade do fundo.

[1] Algoritmo muito usado em imagem de objectos sobre fundos contrastantes. Atribuindo uma etiqueta ao primeiro pixel da imagem e tentando propagar a etiqueta pelos pixéis à sua direita ou abaixo dele. Podem ser usadas adjacências a 4 ou a 8 pixéis.



a) Imagem Original b) Etiquetar pixeis >60 c) Propagação das etiquetas

Figura 4.14 – Componentes Ligados.

O resultado da primeira iteração sobre uma imagem de exemplo pode ser observado na Figura 4.14.b. Neste caso foram etiquetados apenas os pixéis com valor de intensidade maior que 60. Nas iterações seguintes, o algoritmo propaga as etiquetas de menor valor pelos pixéis adjacentes.

O algoritmo procede da seguinte forma para efectuar a etiquetagem de uma imagem:

1. Atribuir uma etiqueta diferente a todos os pixéis não nulos.
2. Da esquerda para a direita, de cima para baixo, atribuir a cada pixel a etiqueta de menor valor de entre a sua e as dos seus vizinhos.
3. Se não houver qualquer propagação de etiquetas, parar.
4. Da direita para a esquerda, de baixo para cima, atribuir a cada pixel a etiqueta de menor valor de entre a sua e as dos seus vizinhos.
5. Se não houver qualquer propagação de etiquetas, parar.
6. Repetir a partir do segundo passo.

Como pode ser observado, na Figura 4.15, o resultado da execução do algoritmo, cria pequenas imagens que serão posteriormente modeladas individualmente.

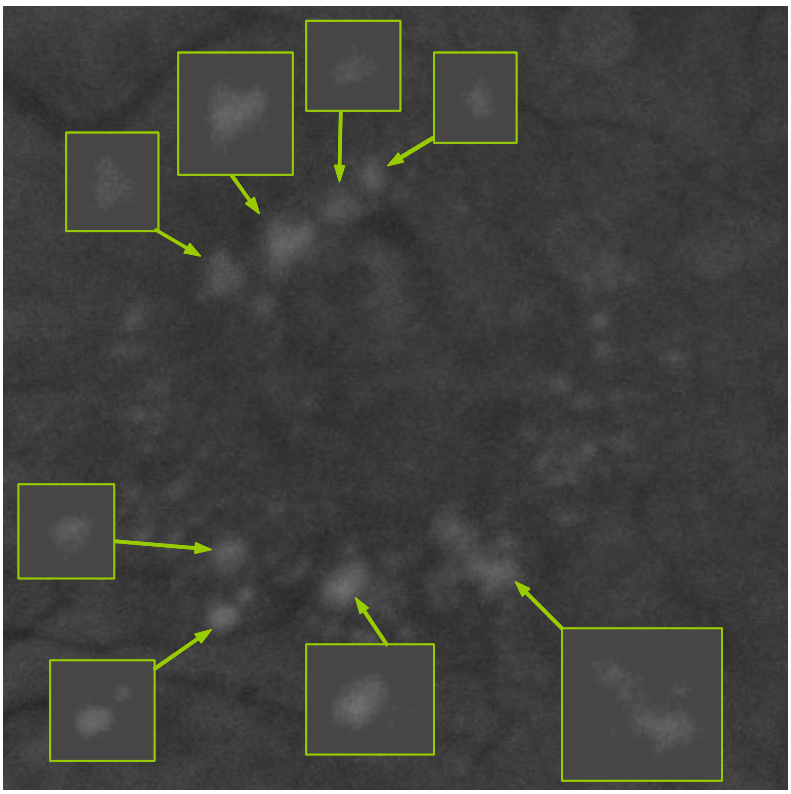


Figura 4.15 – Resultado do algoritmo de agrupamento de Drusas

Após este processo, é guardada a informação da localização de cada nova imagem, para posteriormente ser usada na restituição da imagem.

4.4 Detecção de Drusas

Para uma optimização do processo de detecção e quantificação de drusas optou-se por fornecer a localização aproximada do centro de cada drusa, assim como o valor do máximo local.

Para detectar as drusas foi usado um algoritmo de detecção de máximos locais que tem como uma das suas principais características uma boa imunidade ao ruído. Este é um algoritmo de segmentação de imagem que recorre ao gradiente para efectuar uma etiquetagem da imagem. Em certas imagens de média resolução, é normal haver mais do que um pixel cujo gradiente esteja na direcção de um máximo local. Seguindo este gradiente em cada pixel, será encontrado um máximo local. Este é o princípio de funcionamento do algoritmo utilizado que é muito similar aos algoritmos de etiquetagem de componentes ligados e transformada de *Watershed*[1].

[1] Algoritmo bastante usado para segmentar imagens.

O algoritmo de detecção de máximos locais exposto no artigo [10] é composto por quatro fases distintas:

1. Determinação do gradiente.
2. Propagação de etiquetas.
3. Compatibilização de etiquetas.
4. Junção de etiquetas que por possuírem características compatíveis, podem ser consideradas como pertencentes ao mesmo máximo local.

Na Figura 4.16 podemos ver como se desenvolve o processo de localização dos máximos locais assim como das suas áreas de influência. Em primeiro lugar é calculado o gradiente da imagem usando o operador de Sobel[1] numa janela de 3x3 pixeis, determinando em cada pixel um vector que indica a direcção para o próximo pixel de intensidade superior. A segunda fase consiste em analisar sequencialmente a imagem da esquerda para a direita e de cima para baixo, atribuindo a cada pixel não analisado uma nova etiqueta e propagando-a na direcção do vector do gradiente, até que este indique um pixel já analisado.

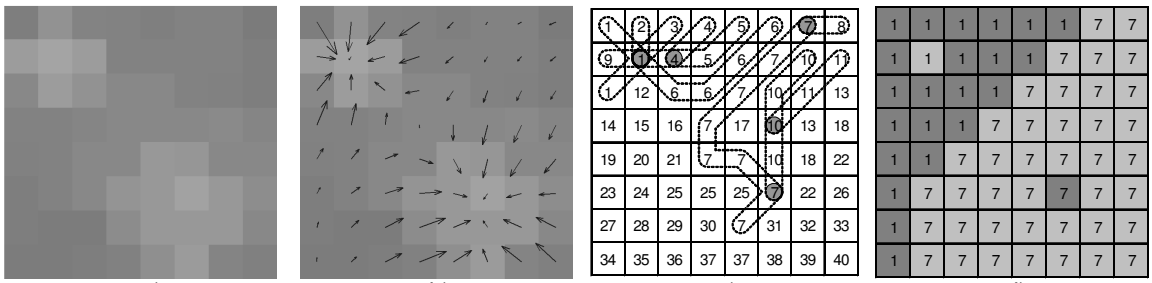


Figura 4.16 – Exemplo do algoritmo de localização de Drusas
a) Imagem original; b) Gradiente da imagem; c) Propagar etiquetas; d) Compatibilização de etiquetas.

A terceira fase consiste em compatibilizar todas as etiquetas que foram identificadas na segunda fase, criando assim uma imagem segmentada em zonas que contribuem para um máximo local. O pixel de maior intensidade de cada região é definido como o máximo local.

Na última fase, todas as regiões vizinhas em que a diferença dos níveis de intensidade da imagem não seja superior a um valor predeterminado são unidas.

[1] O Operador de Sobel é um detector de fronteiras, utilizado para se obter descontinuidades nos tons de cinza da imagem, ou seja, ele permite a determinação de fronteiras na imagem.

Após este processo é guardada a informação da localização de cada Drusa para posteriormente ser usada pelo algoritmo de modelação.

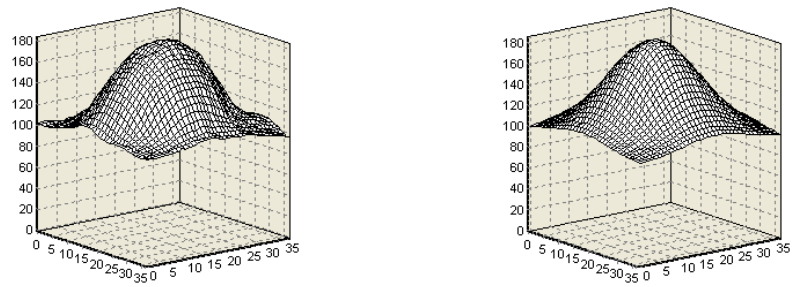
4.5 Modelação de Drusas

A investigação para efectuar a modelação das Drusas foi composta por duas fases. A primeira fase consistiu na pesquisa sobre qual a função matemática que apresentasse mais semelhanças com a forma de uma Drusa. A segunda parte tratou da escolha do algoritmo de optimização.

4.5.1 Função de modelação

De forma a modelar a imagem é importante criar um modelo de cada Drusa detectada, procedeu-se então à escolha de uma função matemática adequada. A função de modelação foi escolhida com base na comparação visual das Drusas.

Nas Figura 4.17.a e Figura 4.17.b pode-se observar a semelhança entre uma Drusa isolada e uma função Gaussiana, respectivamente.



a) Vista 3D imagem original **b)** Vista 3D função Gaussiana

Figura 4.17 – Função de modelação

Devido às semelhanças encontradas entre a imagem da Drusa (Figura 4.17.a) e a função matemática (Figura 4.17.b), foi escolhida a função Gaussiana para modelar as Drusas. A função utilizada encontra-se apresenta-se seguidamente bem como a descrição dos parâmetros constituintes da mesma.

$$G(x,y) = a \times e^{-\left(\frac{X^2}{S_x}+\frac{Y^2}{S_y}\right)^{\frac{2}{d}}} + z_0 \qquad \textbf{Equação 3}$$

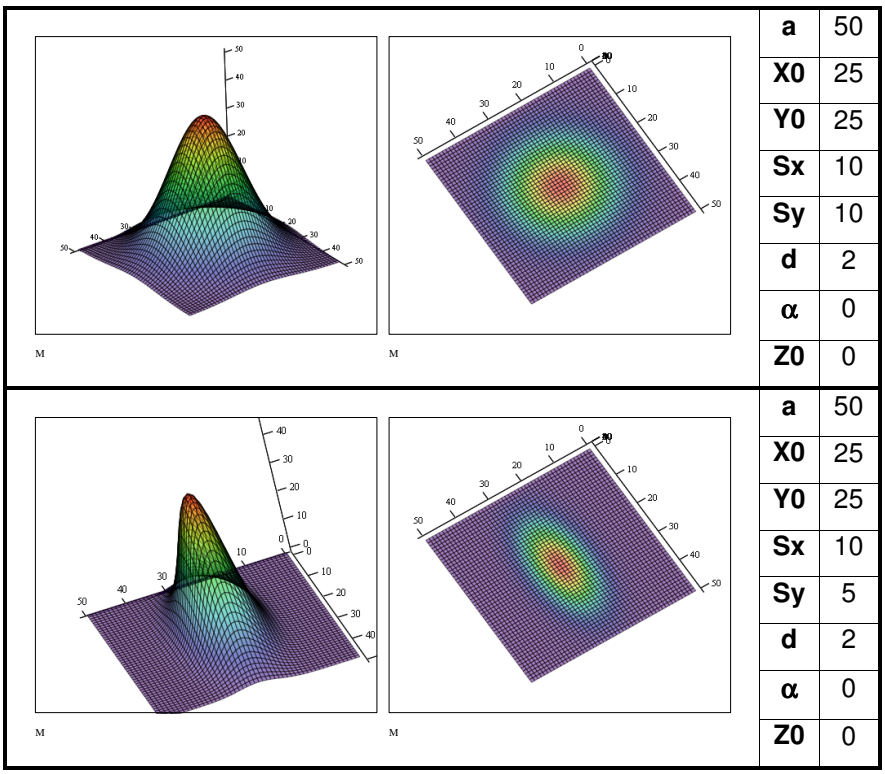
onde,

$$X = (x - x_0).\cos(\theta) + (y - y_0).\sin(\theta)$$
$$Y = -(x - x_0).\sin(\theta) + (y - y_0).\cos(\theta)$$
$$S_y = S_F.S_x$$

Esta função contém nove parâmetros de ajuste, nomeadamente:

a – Amplitude	θ – Rotação
(x_0, y_0) – Coordenadas do centro	*D* – Factor de forma
Z_0 – Valor do fundo	S_F – Factor de escala (S_x / S_y)
S_x – Factor de escala em XX	S_y – Factor de escala em YY

De forma a ajustar a função gaussiana à forma da Drusa é necessário variar os parâmetros da equação. Assim sendo, na figura seguinte, podem ser observados alguns exemplos do resultado destas variações.

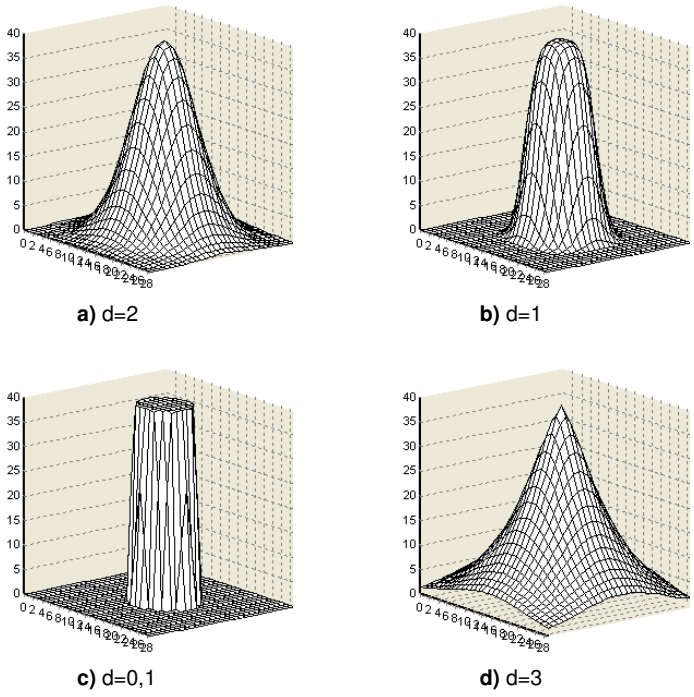


a	50
X0	25
Y0	25
Sx	10
Sy	10
d	2
α	0
Z0	0

a	50
X0	25
Y0	25
Sx	10
Sy	5
d	2
α	0
Z0	0

Figura 4.18 – Exemplos de funções Gaussianas com diversos parâmetros

A equação Gaussiana usada é uma versão modificada da equação típica uma vez que contém um factor de forma **d**, permitindo assim mais um grau de liberdade para a função melhor se ajustar à imagem. Na Figura 4.19 pode-se observar diversos exemplos com diferentes valores de **d**.



a) d=2 **b)** d=1

c) d=0,1 **d)** d=3

Figura 4.19 – Exemplos da função Gaussiana utilizando diversos valores para o parâmetro d.

4.5.2 Algoritmo de Optimização

O algoritmo usado para fazer a optimização dos parâmetros foi o Levenberg-Marquardt [11] que é um algoritmo de optimização de modelos não lineares.

Este algoritmo é uma combinação dos métodos *steepest descent* e Gauss-Newton[1]. Quando a solução actual está longe da solução correcta, o algoritmo comporta-se como o método *steepest descent* (or *Gradient descent*). Caso contrário, quando a solução actual está próxima da solução correcta, comporta-se como o método de Gauss-Newton.

O algoritmo de optimização Levenber-Marquardt tornou-se um standard das rotinas mínimos quadrados de funções não lineares [12]. Para uma função matemática dependente de vários parâmetros, o algoritmo ajusta esses parâmetros de forma a minimizar o erro da função chi-quadrado(χ^2).

[1] Método Gauss-Newton é usado para a solução de sistemas de equações não lineares.

$$\chi^2(a) = \sum_{i=1}^{N} \left[\frac{z_i - z(x_i; y_i; a)}{\sigma_i} \right]^2 \quad \textbf{Equação 4}$$

Esta equação representa a distância entre a solução actual ($z(x_i, y_i, a)$) e a solução correcta (z_i), e o σ representa o desvio padrão.

O algoritmo Levenberg-Marquardt é iterativo sendo inicializado num determinado ponto de partida a_0. Em seguida, o algoritmo produz uma série de vectores a_1, a_2, …., que convergem para um mínimo local.

A diferença entre o primeiro valor do chi-quadrado (χ^2) e o novo valor de chi-quadrado em cada iteração dá uma ideia do sucesso da optimização. Outra variável que mostra o sucesso de cada iteração é o λ, pois esta variável diminui com um factor de dez se o novo valor do chi-quadrado for menor que o anterior. Caso contrário, aumenta com um factor de dez.

Um problema deste algoritmo é a convergência. Para resolver este problema, o algoritmo foi alterado e foram adicionadas duas condições de paragem. A primeira condição impõe que se o valor do chi-quadrado não melhorar em quatro iterações consecutivas o algoritmo pára. A segunda condição diz que se o valor do chi-quadrado chegar a um valor predefinido o algoritmo pára.

5 Aplicação desenvolvida (AD3RI)

Após a investigação feita foi desenvolvida uma aplicação para testes e posterior validação por técnicos especializados.

A aplicação desenvolvida foi baseada num pacote de software já elaborado denominado de *Image Processing Tool*, desenvolvido pelo Mestre André Damas Mora. A nova aplicação chama-se *Automatic Drusens Deposits Detection in Retina Images* (AD3RI). Foi desenvolvida uma aplicação bastante funcional e de utilização simples de forma a testar os algoritmos desenvolvidos. Foi usado o programa C++ Builder para o desenvolvimento da aplicação.

A linguagem de programação usada (C++) é uma linguagem orientada a objectos[1], tendo a aplicação sido desenvolvida com base em várias classes que serão enumeradas seguidamente.

- *ImageClass* – contém diversos métodos para processamento de imagem como por exemplo o filtro de média, mediana, Cmeans, erosão, dilatação, etc..
- *IDSSpots* – classe responsável pelo agrupamento das manchas de Drusas.
- *Gauss* – classe que implementa uma estrutura que suporta a equação *gaussiana*. Contendo diversos métodos para o cálculo da equação gaussiana.
- *Fgauss* – classe complementar da anterior, contendo a função matemática a modelar.
- *MultiImagesProcessing* – contém os métodos necessários para o processamento de várias imagens (após o agrupamento das Drusas, as várias imagens são passadas para esta classe que é responsável pelo seu processamento).
- *Mrqmin* – classe que implementa uma parte do algoritmo Levenber-Marquardt.
- *Mrqcof* – esta classe é complementar da anterior, efectuando o calculo do erro chi-quadrado.

[1] Na programação orientada a objectos, implementa-se um conjunto de classes que definem os objectos do sistema de software. Cada classe determina o comportamento (definidos nos métodos) e estados possíveis (atributos) dos seus objectos, assim como o relacionamento com outros objectos.

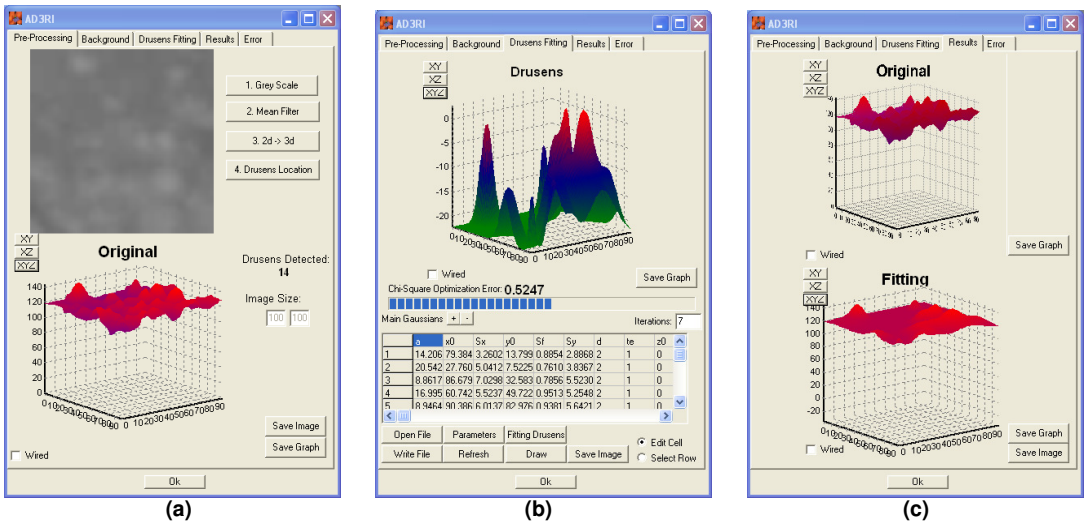
- *SeqWatershed* – contém os métodos necessários para a detecção dos máximos das Drusas.
- *Watershed* – classe complementar da anterior.
- ChartInterface – classe que gere toda a parte funcional e interacção com o utilizador.

As funcionalidades básicas da aplicação incluem:

- Modelação da imagem.
- Correcção da iluminação.
- Detecção automática das Drusas.
- Marcação manual das Drusas a modelar.
- Importação dos parâmetros das funções gaussianas a partir de um ficheiro de texto.
- Exportação dos parâmetros das funções gaussianas para um ficheiro de texto.
- Salvar imagem modelada.
- Salvar gráficos 3D.
- Filtro de redução de ruído.

A aplicação está dividida em cinco painéis principais nomeadamente:

- Preprocessing – onde se encontram a ferramenta de redução de ruído e o algoritmo de localização automática de Drusas.
- Background – onde é efectuada a correcção da iluminação.
- Drusen Fitting – onde as Drusas são modeladas.
- Results – onde é visualizado o resultado da optimização.



(a) (b) (c)

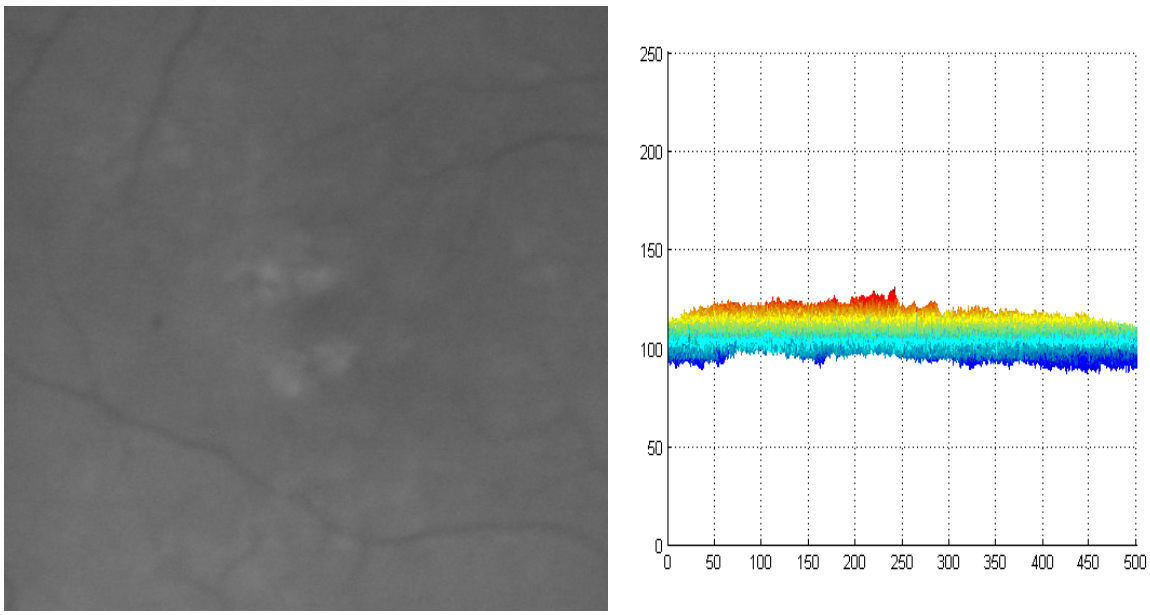
Figura 5.1 – Conjunto de imagens da aplicação desenvolvida

Os valores dos parâmetros das funções Gaussianas estão dispostos numa tabela, podendo ser introduzidos automaticamente (usando os valores determinados pelo algoritmo de detecção automática de Drusas), manualmente ou podem ser carregados a partir de um ficheiro de texto. A Figura 5.1.b mostra o painel Drusens Fitting onde pode ser vista uma tabela com os parâmetros das funções gaussianas, assim como algumas ferramentas associadas ao procedimento de modelação. O utilizador tem a possibilidade de salvar a vista 3D ou 2D da imagem após a modelação.

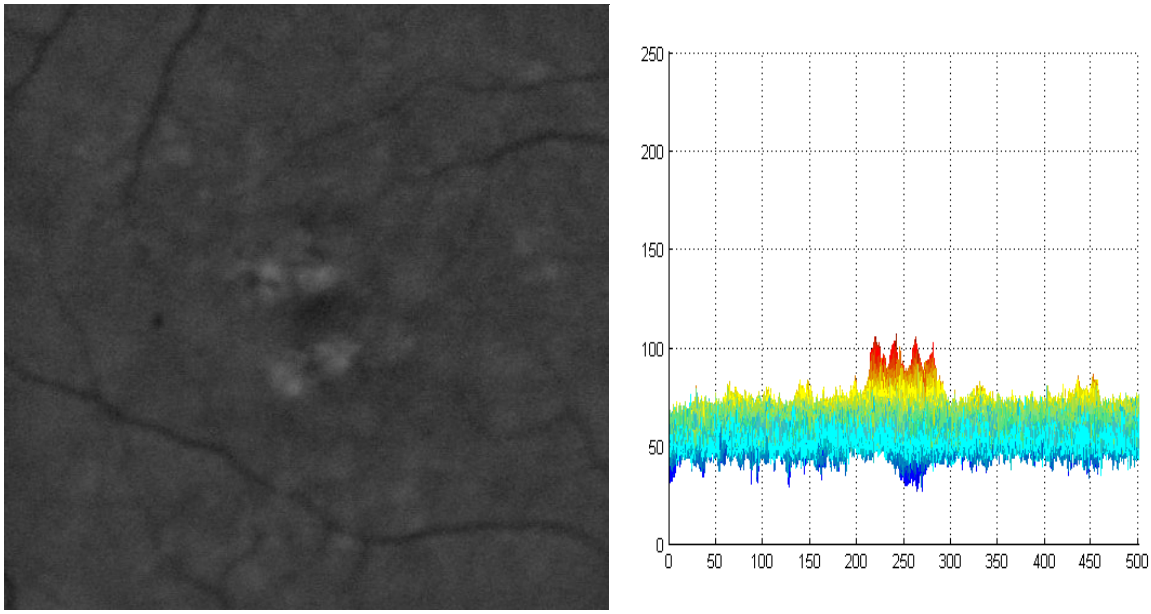
6 Resultados

A validação dos algoritmos desenvolvidos foi efectuado tendo como base imagens obtidas através da técnica de fotografia de fundus analógica, digitalizadas usando um scanner. Embora as imagens obtidas terem aproximadamente 1600x1100 pixéis, de forma a reduzir o tempo de processamento foi apenas considerada uma área de interesse com cerca de 540x540 pixel na zona da mácula.

O primeiro algoritmo a ser avaliado foi o algoritmo de normalização do fundo previamente explicado na secção 4.2.1.



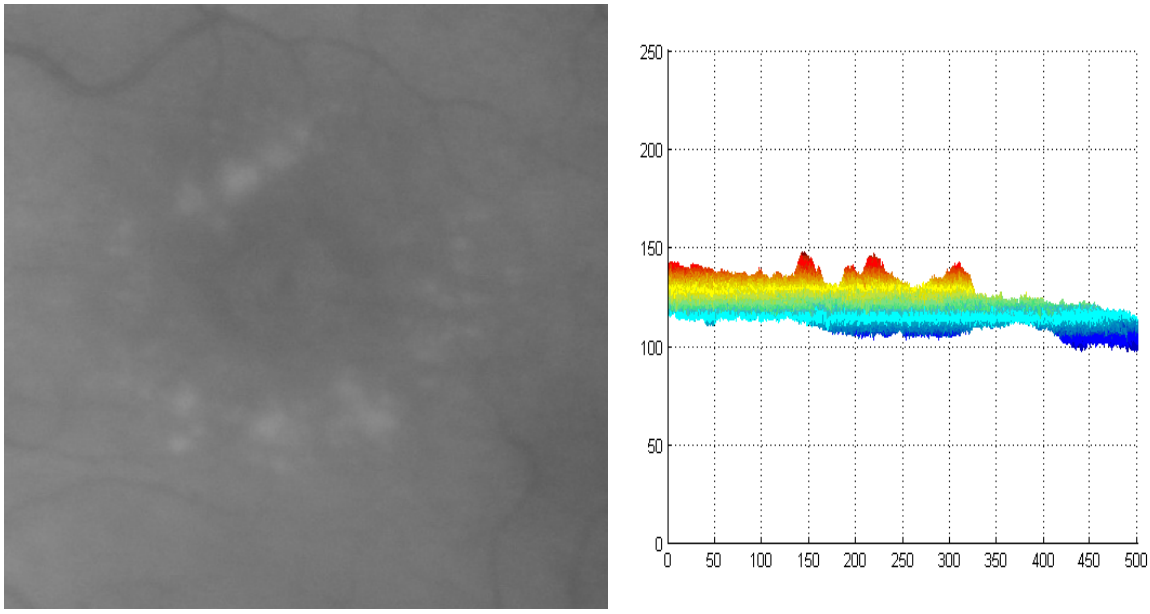
a) Imagem Original



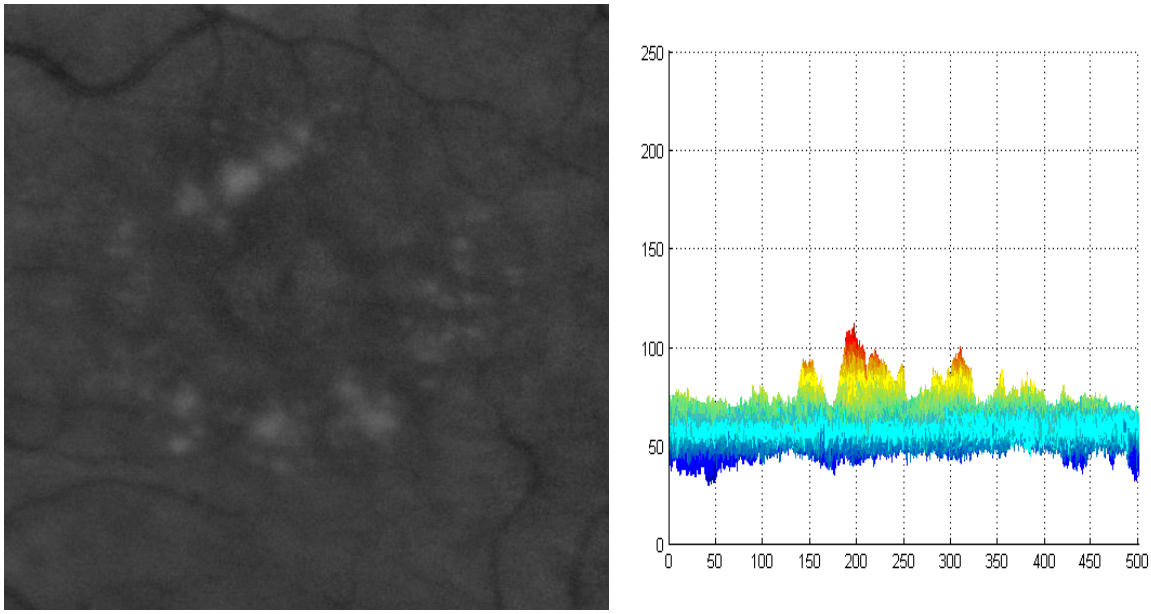
b) Imagem Normalizada

Figura 6.1 – 1º Exemplo de Normalização do Fundo

No primeiro exemplo de normalização do fundo, que pode ser observado na Figura 6.1.b, verifica-se que observar-se que os valores dos pixéis estão mais dispersos que na imagem original, ou seja, o contraste da imagem foi aumentado, não perdendo no entanto informação.



a) Imagem Original

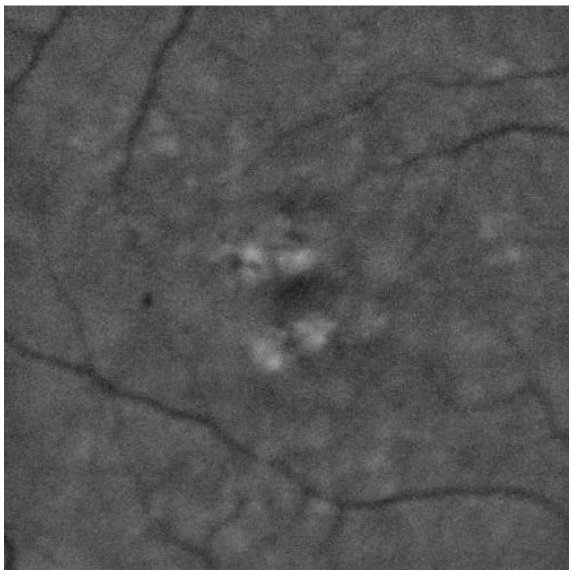


b) Imagem Normalizada

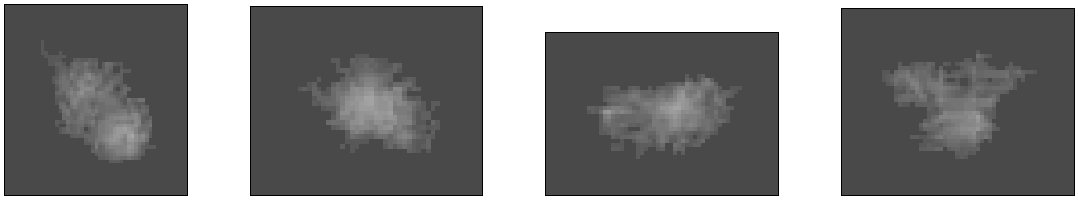
Figura 6.2 – 2º Exemplo de Normalização do Fundo

Na Figura 6.2 foi conseguido também o objectivo pretendido, que era efectuar a normalização da imagem, centrando-a em torno do ponto 50 e aumentando o contraste. A uniformização das imagens foi também alcançada com sucesso.

De seguida procede-se à divisão da imagem normalizada em pequenas imagens, que representam agrupamentos de manchas de Drusas.



a) Imagem Normalizada

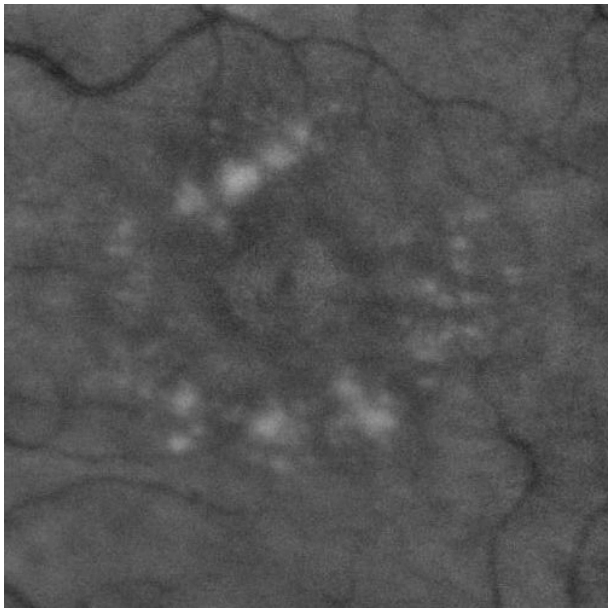


b) Grupos de manchas Drusas

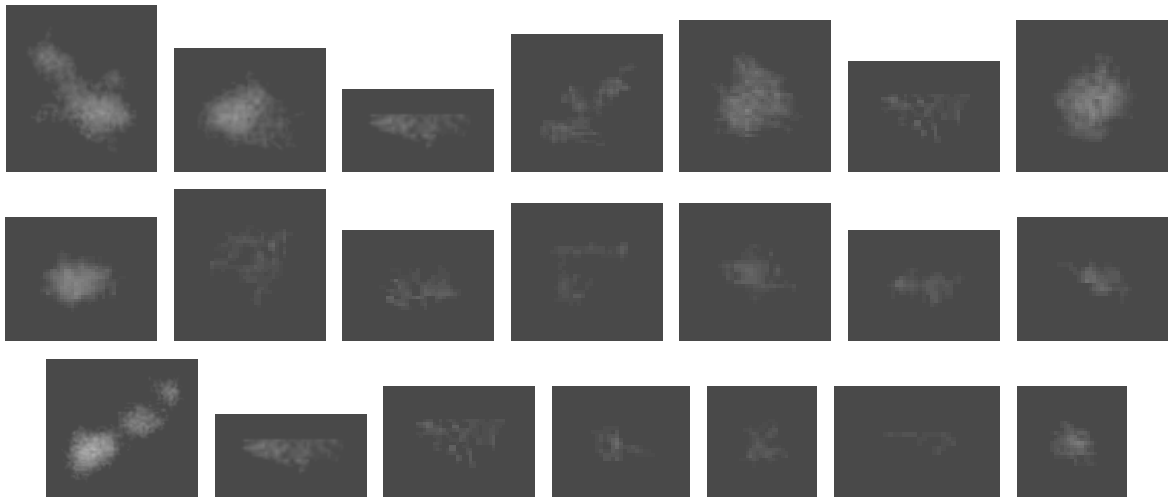
Figura 6.3 – 1º Exemplo de Agrupamento de Drusas.

Na Figura 6.3 encontra-se o primeiro exemplo da aplicação do algoritmo de agrupamento de Drusas. Foram apenas apresentadas as imagens mais significativas, pois na realidade o algoritmo para a imagem apresentada gerou 24 novas imagens sendo elas posteriormente modeladas. No entanto devido à sua insignificância, muitas delas são posteriormente rejeitadas pelo algoritmo de quantificação, não sendo tidas em consideração para o resultado final.

Na imagem da Figura 6.4 encontra-se o segundo exemplo do algoritmo de agrupamento de Drusas. Pela mesma razão apenas foram apresentadas algumas imagens do resultado. O algoritmo gerou 117 sub-imagens. Executando o algoritmo de quantificação de Drusas foram consideradas possíveis Drusas apenas 50.



a) Imagem Normalizada



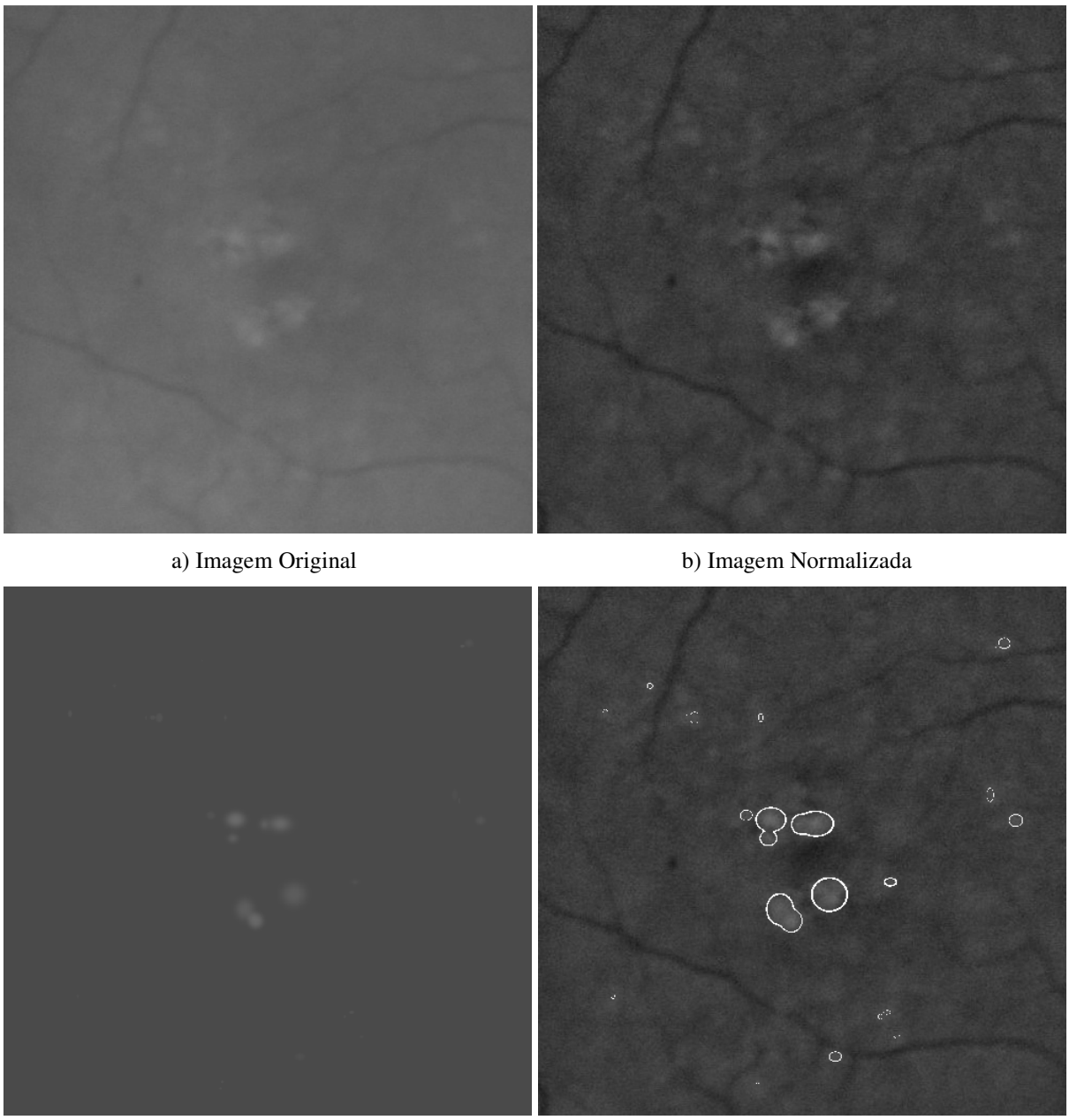
b) Grupos de manchas Drusas

Figura 6.4 – 2º Exemplo de Agrupamento de Drusas.

De seguida o algoritmo de optimização efectua a modelação de todas as imagens resultantes do algoritmo anterior (agrupamento de Drusas).

De modo a modelar a primeira imagem (Figura 6.5) o algoritmo optimizou 24 imagens demorando pouco mais que 1 minuto. Este tempo é tido em consideração pois de futuro espera-se que a aplicação seja utilizada por médicos oftalmologista, sendo, portanto, importante que o processamento seja feito em tempo útil.

Tendo em conta os algoritmos anteriores este tempo de processamento (1 minuto) é considerado aceitável, embora tendo em conta que a imagem fosse simples.



a) Imagem Original b) Imagem Normalizada

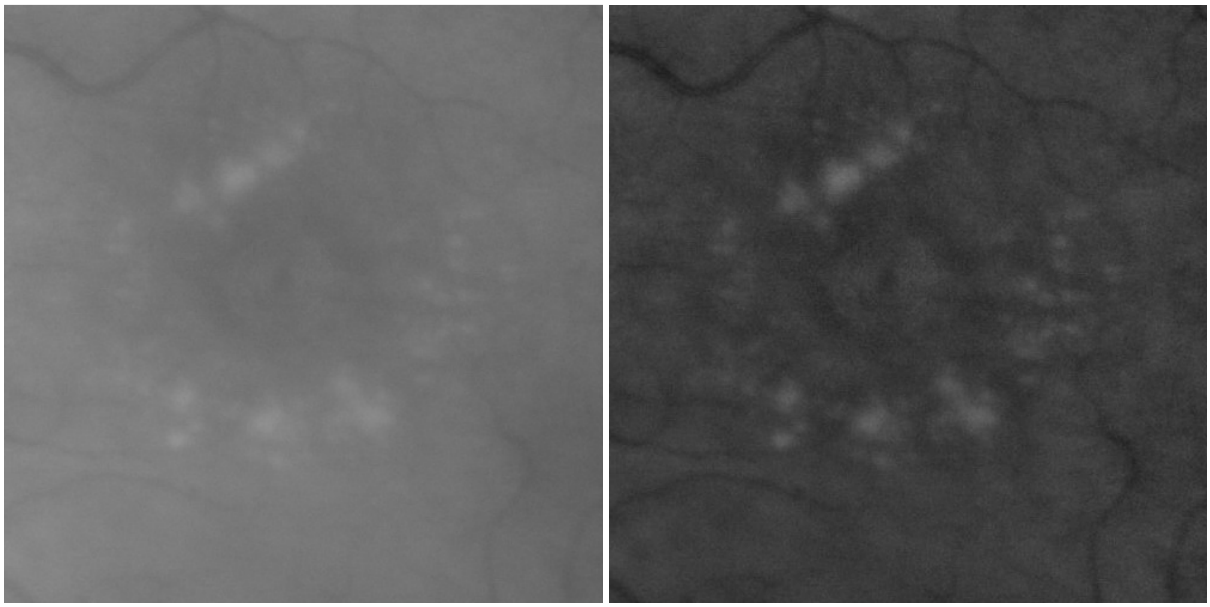
c) Imagem Optimizada d) Demarcação das áreas das Drusas

Figura 6.5 – 1º Exemplo de Optimização de Drusas.

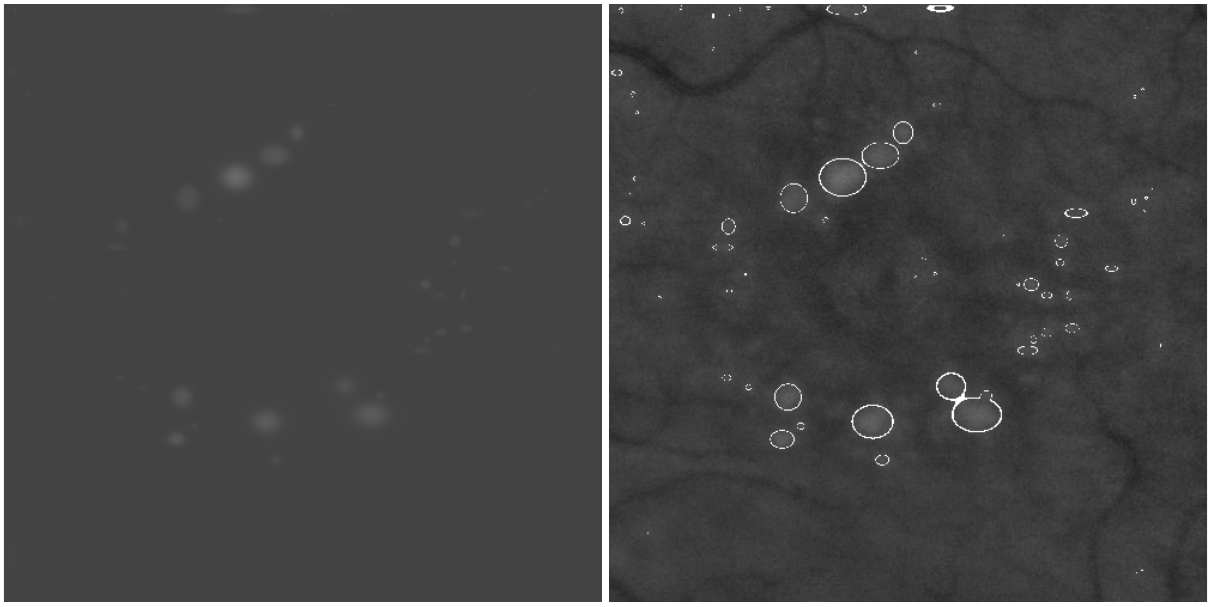
A Figura 6.6 apresenta o segundo exemplo da execução do algoritmo de optimização. Nesse caso o resultado alcançado pode ser considerado aceitável, visto o algoritmo ter optimizado 117 imagens em menos de 5 minutos. Em relação à imagem resultante, observa-se que todas as imperfeições desapareceram optimizando apenas as Drusas com maior relevância.

Os resultados das optimizações são guardados não só em imagens mas também os parâmetros das funções Gaussianas que modelaram as Drusas são guardados de modo a posteriormente ser possível efectuar outros cálculos, tais como áreas, volumes, etc.. Esse facto torna também possível a comparação

posterior com outras imagens tiradas em diferentes alturas, podendo assim efectuar-se a avaliação da evolução da anomalia.



a) Imagem Original b) Imagem Normalizada



c) Imagem Optimizada d) Demarcação das áreas das Drusas

Figura 6.6 – 2º Exemplo de Optimização de Drusas.

7 Conclusões

O uso da modelação como auxiliar à detecção de drusas tem como primeira dificuldade a eliminação do ruído e de artefactos provenientes de métodos de compressão. Retirando estas anomalias pode-se assim trabalhar com uma imagem apenas com manchas, reduzindo assim a hipótese de identificação de falsas drusas. Assim sendo, a aplicação desenvolvida é uma ferramenta de apoio à comunidade oftalmológica para a detecção mais precisa e determinística das manchas de Drusas. Esta aplicação possibilita também a quantificação do número de Drusas bem como a sua área oferecendo, deste modo, a possibilidade do clínico efectuar uma avaliação, do desenvolvimento da anomalia, mais concreta com base em valores e não apenas na sua observação.

O algoritmo Levenberg-Marquardt usado para a modelação das drusas, provou adaptar-se e mostrar resultados aceitáveis de modo a cumprir o objectivo proposto.

O algoritmo de agrupamento das Drusas obteve bons resultados e o seu uso tornou claro que o processamento individual de cada Drusa é mais eficiente que o processamento conjunto. Outra vantagem do uso deste algoritmo é que, deste modo, o processamento das imagens das Drusas pode ser distribuído por várias máquinas, sendo de seguida apenas recebido o resultado final do processamento efectuado.

A comparação dos resultados obtidos pelo algoritmo com os resultados obtidos pelos clínicos não é viável, devido à variabilidade encontrada nas análises efectuadas pelos vários clínicos envolvidos no projecto.

Embora o objectivo proposto tenha sido alcançado existe ainda muito a aperfeiçoar. Como perspectivas futuras, está em fase de investigação a optimização do algoritmo de modelação de forma a acelerar o processo de modelação das Drusas, pois embora o algoritmo seja rápido em imagens com poucas Drusas, em imagens com elevado número de Drusas o tempo de processamento aumenta significativamente, tornando difícil a possibilidade da utilização por profissionais.

Fora do âmbito deste projecto encontra-se em investigação um método para isolar as Drusas e suas áreas de influência de modo a minimizar o tempo de processamento e aumentar a precisão.

8 Publicações

F. Moitinho, A. Mora, P. Vieira e J. Fonseca – AD3RI a Tool for Computer – Automatic Drusens Detection.

Apresentado na conferência compIMAGE 2006 (Computational modelling of object represented in images) (21-10-2006)

F. Moitinho, A. Mora, P. Vieira e J. Fonseca – IMAGE SEGMENTATION FOR DRUSEN SPOTS DETECTION AND MODELLING.

Apresentada na conferência CIMED 2007 (27-7-2007)

F. Moitinho, A. Mora, P. Vieira e J. Fonseca – A Drusen Volume Quantification Method based on a Segmentation Algorithm.

Apresentada na conferência VipIMAGE 2007 (I ECCOMAS THEMATIC CONFERENCE ON COMPUTATIONAL VISION AND MEDICAL IMAGE PROCESSING) (17-10-2007)

Mora, A., F. Moitinho, P. Vieira, and J. Fonseca – Using mathematical modelling techniques for drusen quantitative evaluation on retinography examination. Physiological Measurement.

Submetido ao IOP Journal (04-01-2008)

9 Bibliografia

1. Pauleikhoff, D., et al., *Drusen as risk factors in age-related macular disease.* Am J Ophthalmol, 1990. 109(1): p. 38-43.
2. William R. Freeman, M.M.E.-B., MD; Daniel J. Plummer, PhD and *Scanning Laser Entoptic Perimetry for the Detection of Age-Related Macular Degeneration* Archives of Ophtalmology, 2004. 122(11).
3. Peli, E. and M. Lahav, *Drusen Measurement from Fundus Photographs Using Computer Image Analysis*, in *Ophtalmology*. 1986. p. 1575-1580.
4. Sebag, M., E. Peli, and M. Lahav, *Image analysis of changes in drusen area*, in *Acta Ophtalmologica*. 1991. p. 603-610.
5. Phillips, R.P., et al., *Quantification of diabetic maculopathy by digital imaging of the fundus.* Eye, 1991. 5 (Pt 1): p. 130-7.
6. Kirkpatrick, J.N.P., et al., *Quantitative image analysis of macular drusen from fundus photographs and scanning laser ophthalmoscope images.* Eye (Royal College of Ophthalmologists), 1995. 9: p. 48-55.
7. Thdibaoui, A., A. Rajn, and P. Bunel. *A fuzzy logic approach to drusen detection in retinal angiographic images*. in *15th International Conference on Pattern Recognition*. 2000. Barcelona, Spain.
8. Sbeh, Z.B., et al., *A New Approach of Geodesic Reconstruction for Drusen Segmentation in Eye Fundus Images*, in *IEEE TRANSACTIONS ON MEDICAL IMAGING*. 2001. p. 1321-1333.
9. Sbeh, Z.B., et al. *An adaptive contrast method for segmentation of drusen*. in *International Conference on Image Processing*. 1997.
10. Mora, A., P. Vieira, and J. Fonseca. *Drusen Deposits on Retina Images: Detection and Modeling*. in *MEDSIP-2004*. 2004. Malta.
11. Marquardt, D.W., *An algorithm for least-squares estimation of non-linear parameters.* Journal of the Society for Industrial and Applied Mathematics, 1963. 11(2): p. 431-441.
12. William H. Press, S.A.T., William T. Vetterling, Brian P. Flannery, *Numerical Recipes in C*. Second Edition ed. 1999: Cambridge University Press.
13. Xu, C. and J.L. Prince, *Snakes, Shapes, and Gradient Vector Flow*, in *IEEE TRANSACTIONS ON IMAGE PROCESSING*. 1998. p. 359-369.
14. Barthes, A., et al., *Mathematical morphology in computerized analysis of angiograms in age-related macular degeneration.* Med Phys, 2001. 28(12): p. 2410-9.
15. Smith, R.T., et al., *A Method of Drusen Measurement Based on the Geometry of Fundus Reflectance.* BioMedical Engineering OnLine, 2003. 2(10).
16. http://www.esec-passos-manuel.rcts.pt/crdv/baixa.htm#n2
17. http://en.wikipedia.org/wiki/Fundus_camera

Anexo I – Manual da Aplicação

10 Manual da Aplicação

A aplicação de desenvolvida foi baseada num pacote de software já elaborado que dá pelo nome de Image Processing Tool desenvolvida pelo Mestre André Damas Mora.

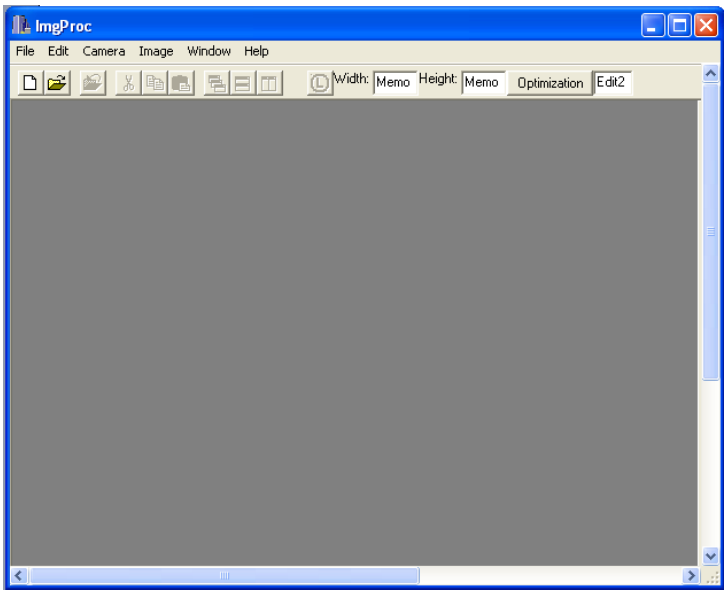


Figura 10.1 – Aplicação Image Processing Tool

Abrir Imagem

A imagem a processar pode ser em formato JPEG (*.jpg) ou Bitmap (*.bmp).

Temos duas formas de abrir um ficheiro de imagem, no botão colocado na toolbar, ou através do menu File.

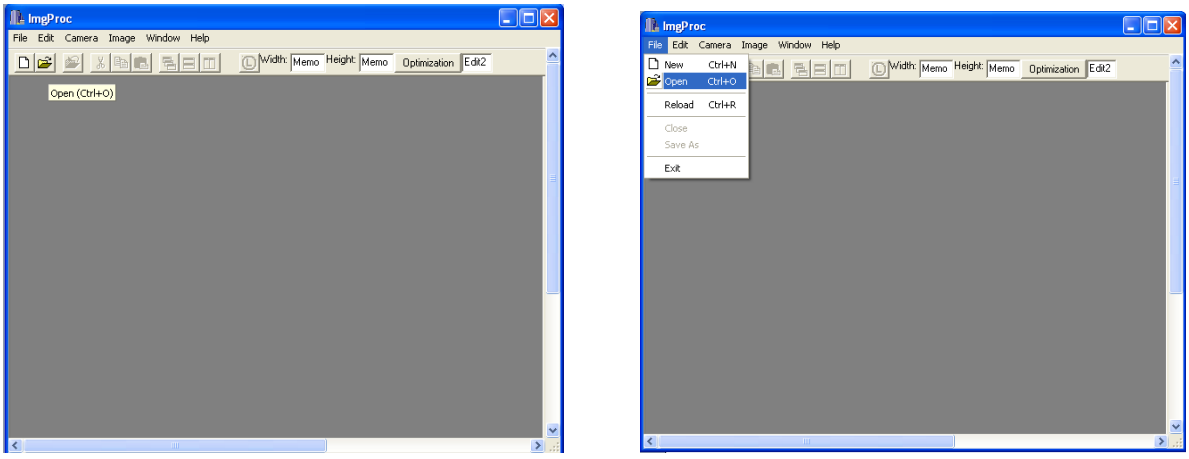


Figura 10.2 – Formas de abrir um ficheiro de imagem

Seguidamente é escolhida a imagem que se deseja modelar.

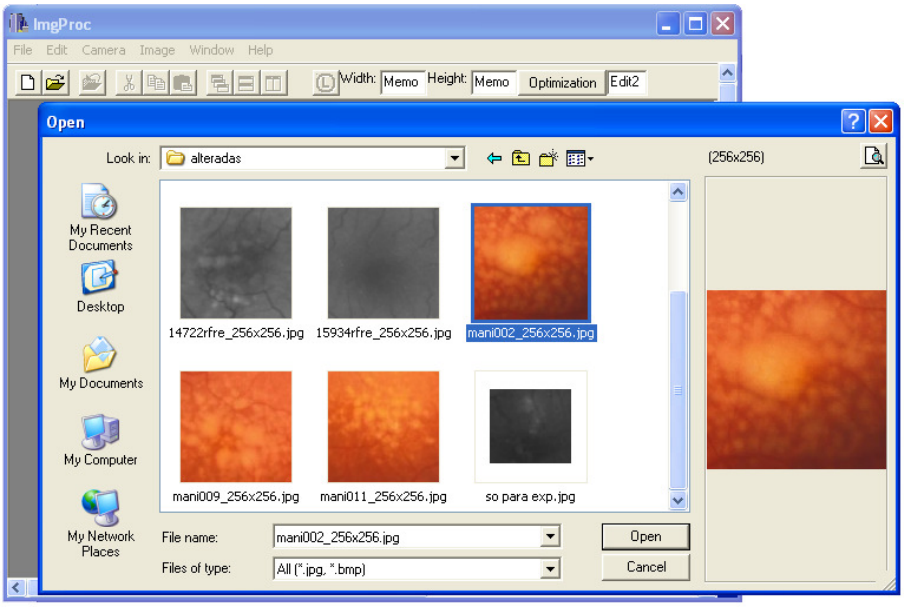


Figura 10.3 – Escolha da Imagem

Abrir a aplicação de Fitting

A aplicação só pode ser iniciada após a abertura do ficheiro de imagem. Para executar a aplicação basta ir ao menu Image e seleccionar Drusens Analysis.

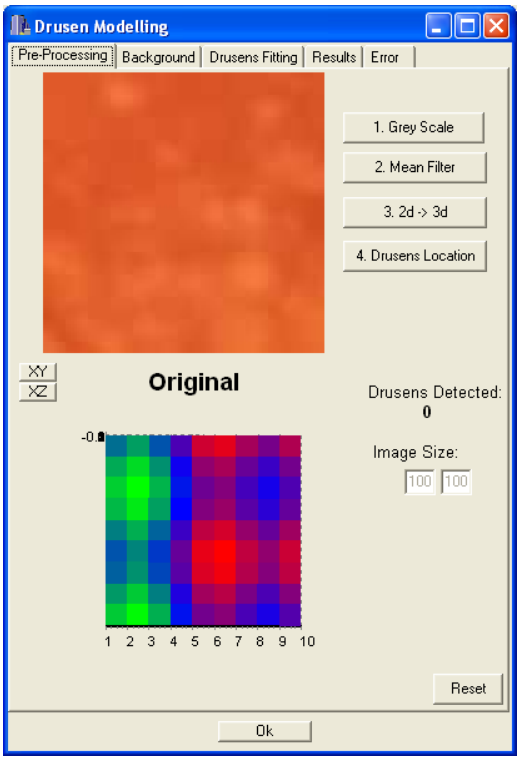


Figura 10.4 – Aplicação de Fitting

O Interface da Aplicação é composto por cinco Menus: Pré-Processing, Background, Drusens Fitting, Results e Error. Procederemos seguidamente à apresentação de cada menu.

Pre-Processing

Este menu inicial de pré-processamento da imagem tem quatro botões principais, que estão numerados, devendo a sua ordem ser respeitada.

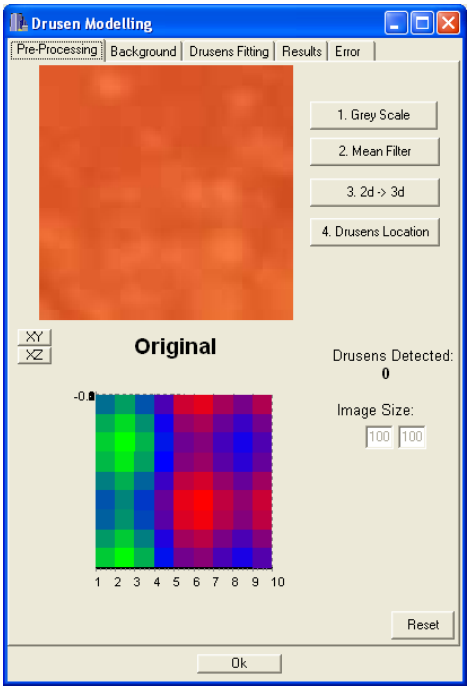


Figura 10.5 – Vista do Form Pré-Processing

1. **Grey Scale**: este botão só se encontra activo se a imagem não estiver em escala de cinzentos. Caso contrário, o utilizador deverá premir este botão de forma a ficar com a imagem em escala de cinzentos.

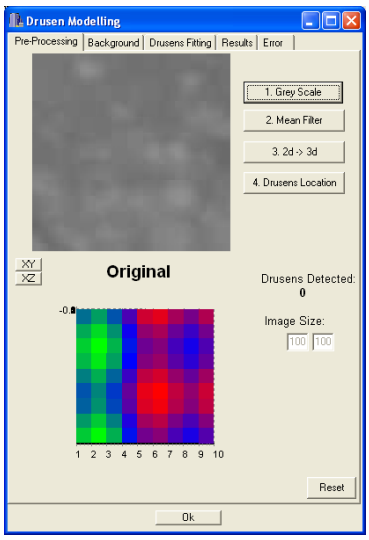


Figura 10.6– Execução do botão nº1

2. **Mean Filter**: este botão executa a passagem de um filtro de média pela imagem, por forma a suavizar os picos indesejados, assim como retirar os efeitos causados por imagens em formato JPG.

3. **2D→3D**: este botão faz passar a imagem que está em 2D para 3D, aparecendo a mesma no gráfico em baixo.

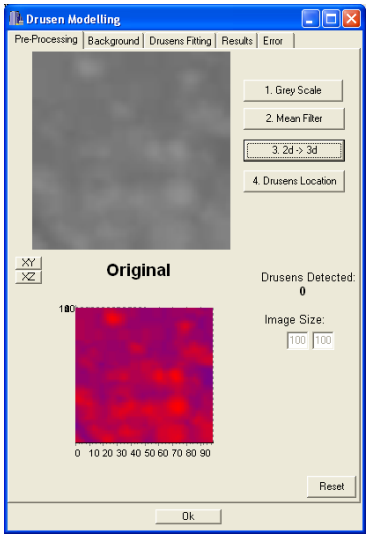


Figura 10.7– Execução do botão nº3

4. **Drusens Location**: Este botão executa o comando de localização de drusas, indicando a quantidade de possíveis drusas encontradas. Como se pode ver no exemplo apresentado na Figura 10.8 temos 14 drusas encontradas.

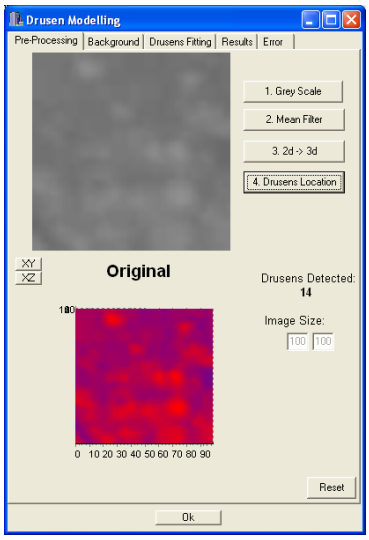


Figura 10.8– Execução do botão nº4

Os botões que se encontram no canto superior esquerdo do gráfico servem para mudar as vistas do gráfico. Na Figura 10.9 podemos ver a diferença entre as duas vistas.

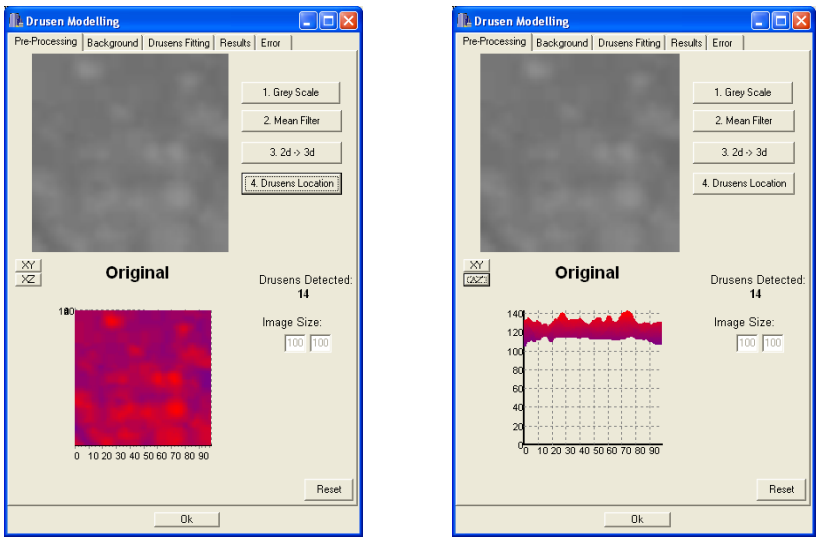


Figura 10.9 – Diferentes vistas XY e XZ

Background

É neste menu que é efectuada a optimização do background, assim como a sua remoção.

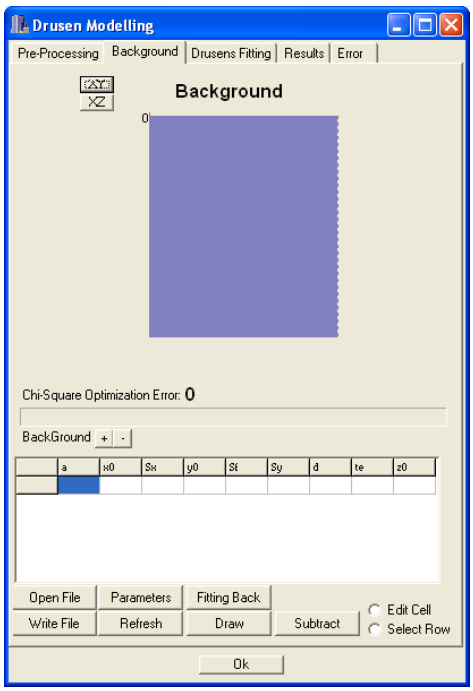


Figura 10.10 – Vista do Form Background

Neste menu podemos ver o background numa tabela onde são indicadas as drusas a optimizar juntamente com um conjunto de diversas funcionalidades. Para adicionar drusas existem três formas diferentes:

1. Premir o botão **Open File** para adicionarmos os parâmetros a partir de um ficheiro de texto, como exemplificado na Figura 10.11.

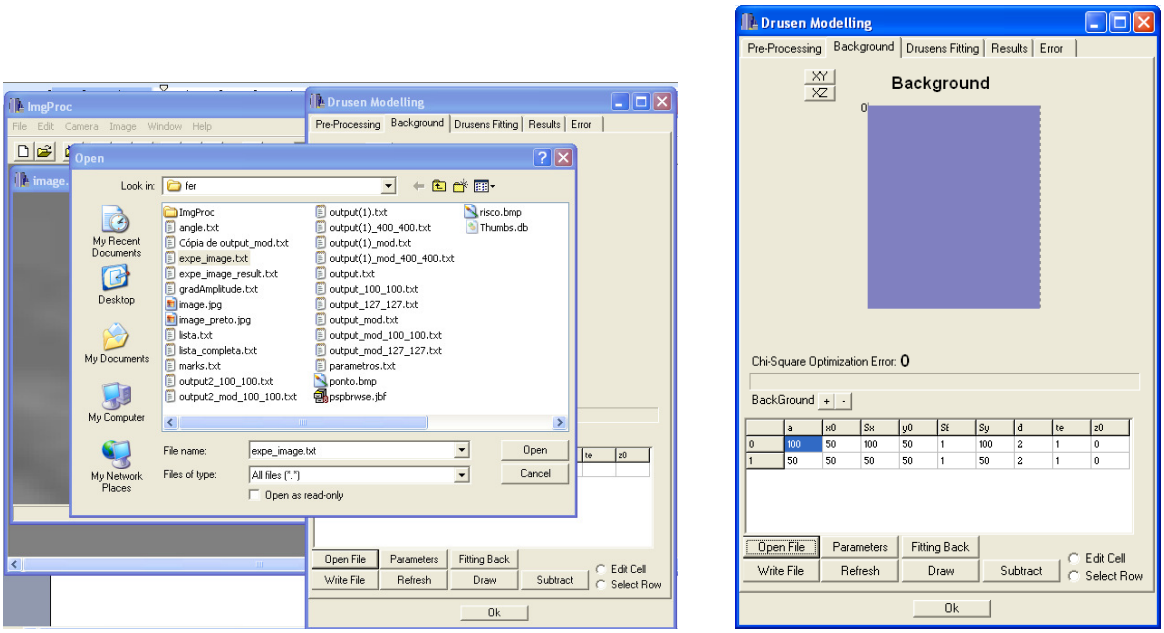


Figura 10.11– Procedimento para escolher um ficheiro que contém parâmetros

2. Premir o botão **Parameters** que coloca valores predefinidos tendo em conta o tamanho da imagem.

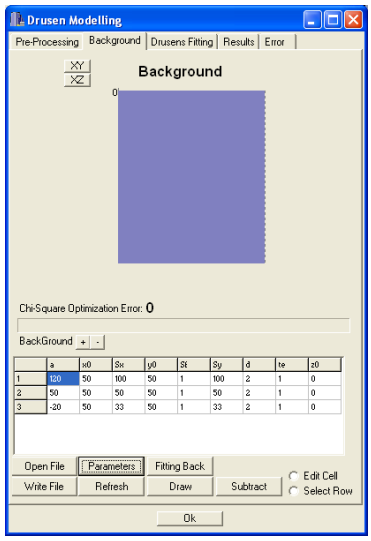


Figura 10.12 – Parâmetros predefinidos para a imagem em questão

3. Inserir os valores manualmente premindo o botão + (mais), para adicionar drusas ou o botão - (menos) para eliminar.

Após cada modificação que se efectue na tabela é necessário premir o botão **Refresh,** para essas alterações tomarem efeito.

De seguida executa-se o comando **Fitting Back** que procede à modelação do background. Aquando deste processo a barra do Chi-Square vai decrescendo.

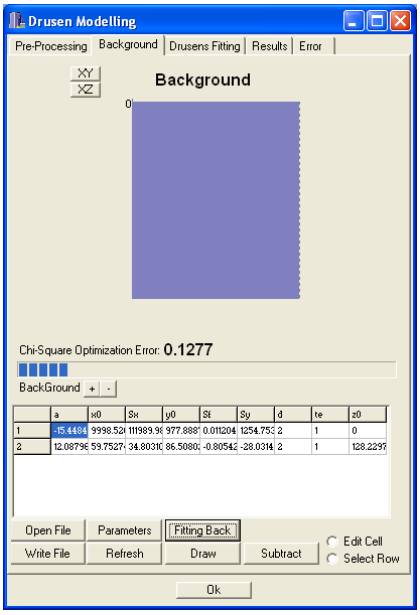


Figura 10.13– Após a execução da optimização

De seguida é necessário premir o botão **Subtract** para a subtracção do background, como podemos ver na Figura 10.14 após a subtracção e rodando o gráfico para termos uma melhor visualização.

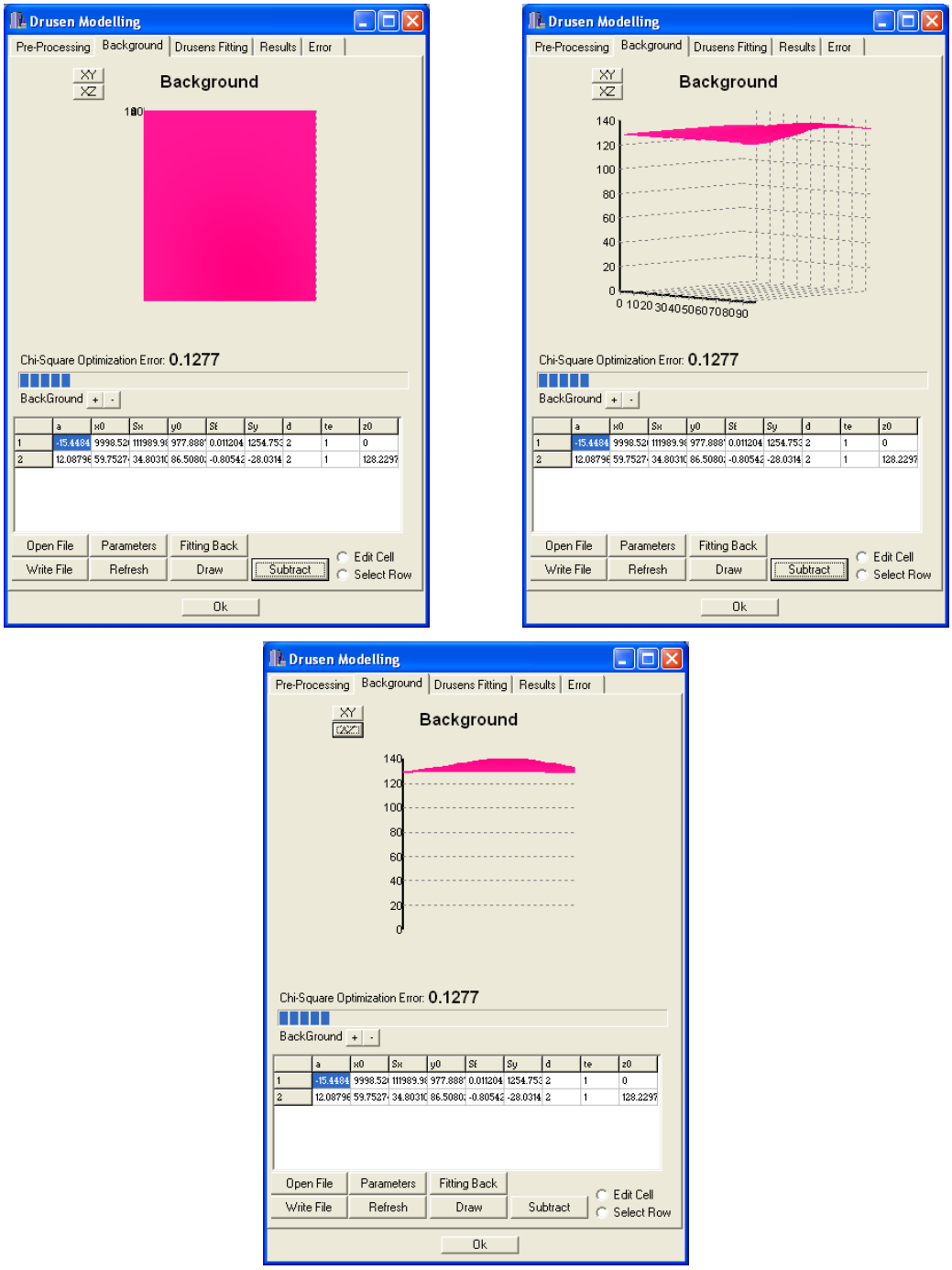


Figura 10.14 – Após a subtracção do Background

Se desejar pode ir ao menu **Pré-Processing** e visualizar a imagem original após a remoção do background. Podemos comprovar a sua remoção vendo a escala em ZZ. Neste caso o maior valor que toma agora é 20, quando anteriormente era 140 como podemos constatar na Figura 10.15.

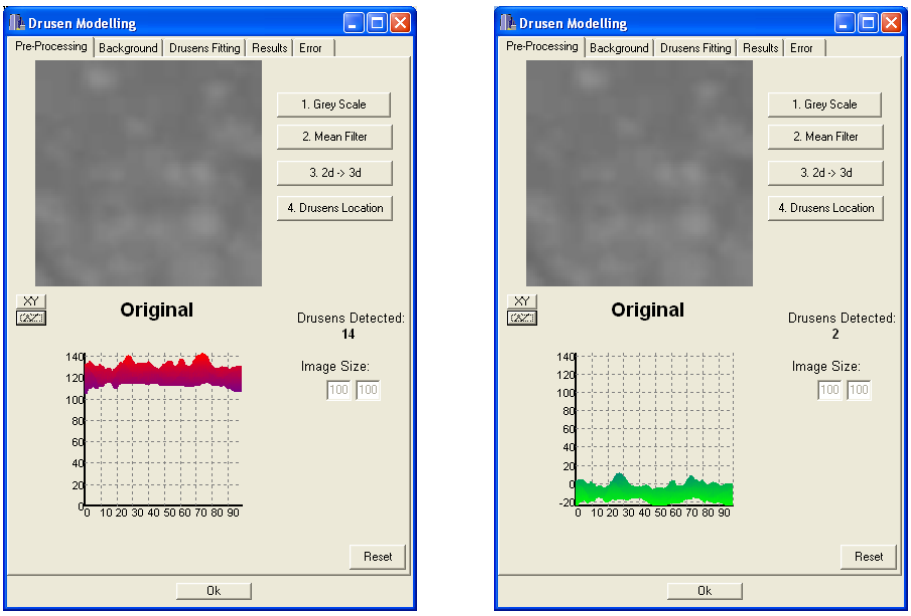


Figura 10.15 – Antes da remoção do background e após a remoção

Existe ainda a possibilidade de guardar os parâmetros em ficheiro de texto após a optimização. Basta para isso premir o botão **Write File** e escolher o nome do ficheiro. Esta funcionalidade torna-se útil para posterior uso dos mesmos parâmetros.

Drusens Fitting

É este menu que nos dá a oportunidade de optimizar as drusas.

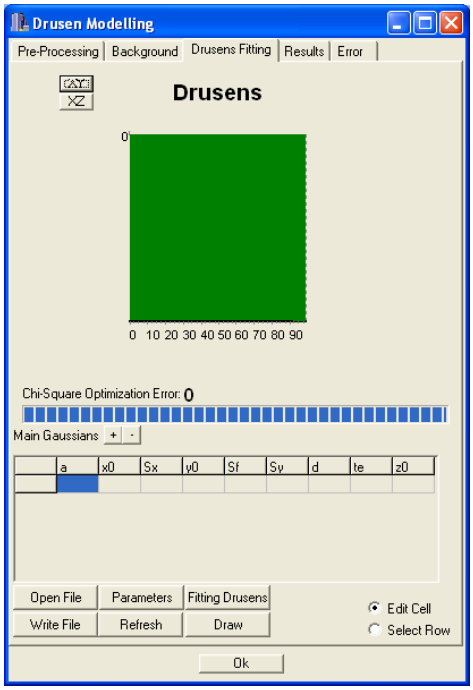


Figura 10.16 – Vista do Form Drusens Fitting

Tal como no menu anterior, existem três formas de inserir parâmetros:

1. Premir o botão **Open File** para adicionarmos parâmetros a partir de um ficheiro, como exemplificado na Figura 10.17.

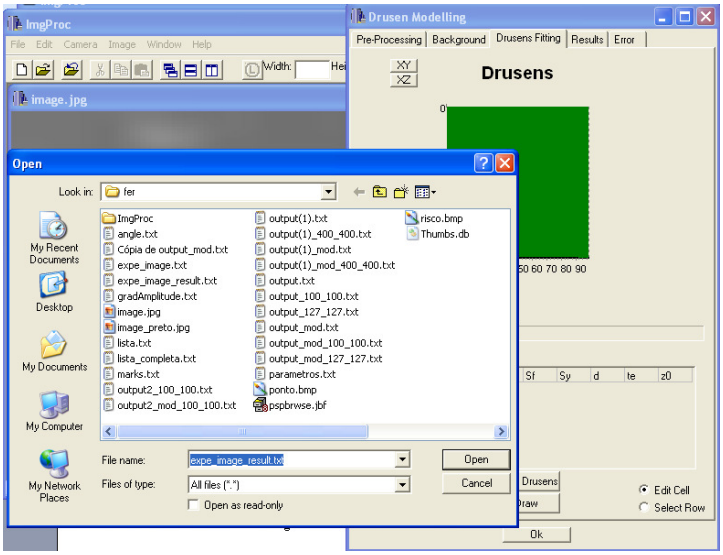


Figura 10.17 – Procedimento para escolher um ficheiro que contém parâmetros

2. Premir o botão **Parameters** que coloca valores predefinidos, fornecidos pelo algoritmo do gradiente.

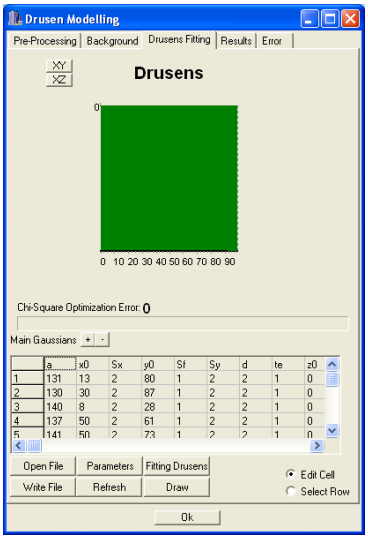


Figura 10.18 – Parâmetros fornecidos pelo algoritmo do gradiente

3. Inserir os valores manualmente premindo o botão + (mais), para adicionar drusas ou o botão - (menos) para eliminar.

Após cada modificação que se efectue na tabela é necessário premir o botão **Refresh,** para essas alterações tomarem efeito.

A optimização das drusas pode ser feita premindo o botão **Fitting Drusens**, de forma semelhante ao background. O valor de chiquadrado vai assim diminuindo até atingir uma das condições de paragem.

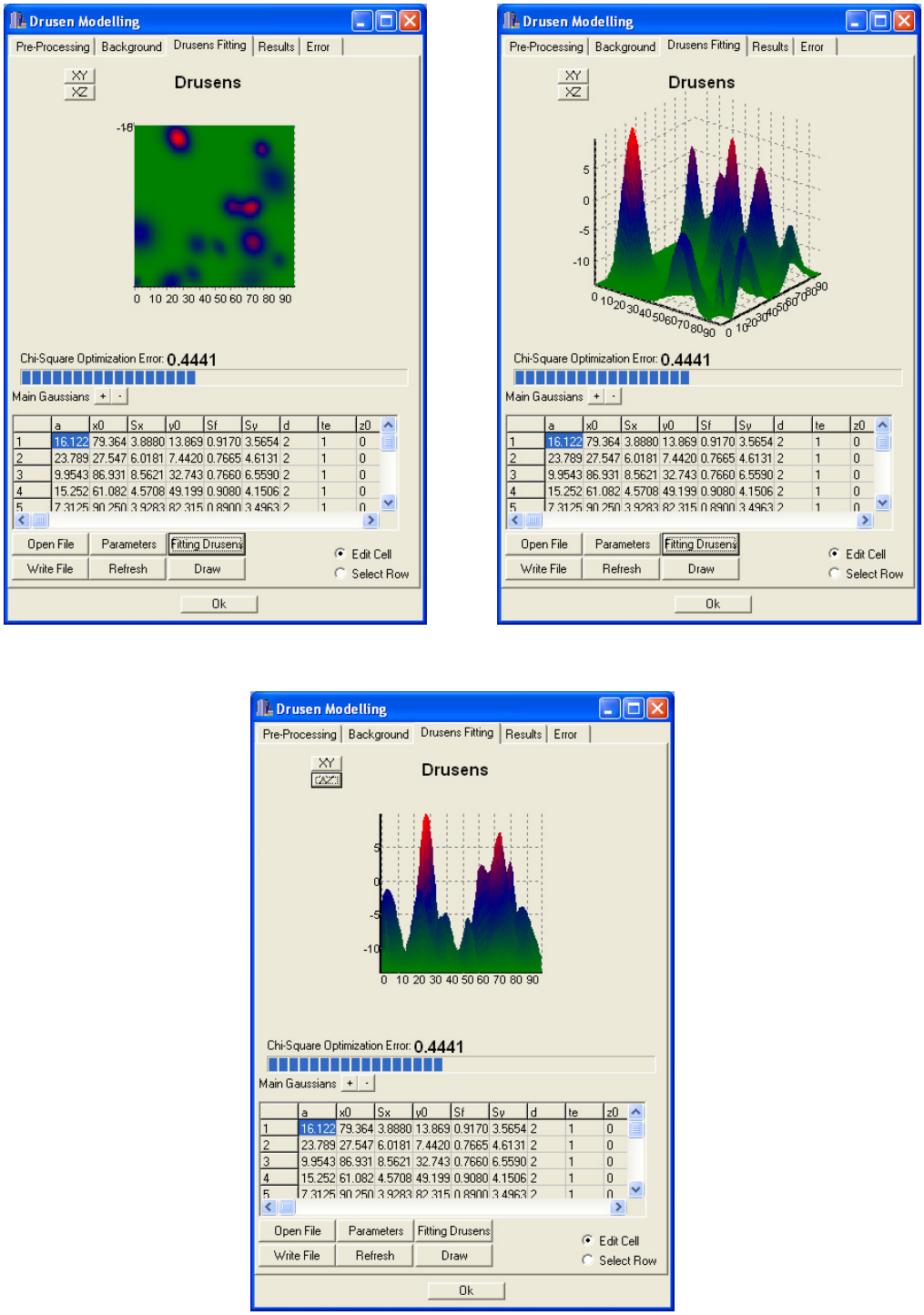


Figura 10.19 – Após a optimização das drusas

Results

Neste menu podemos ver o resultado da optimização e comparar visualmente com a imagem original.

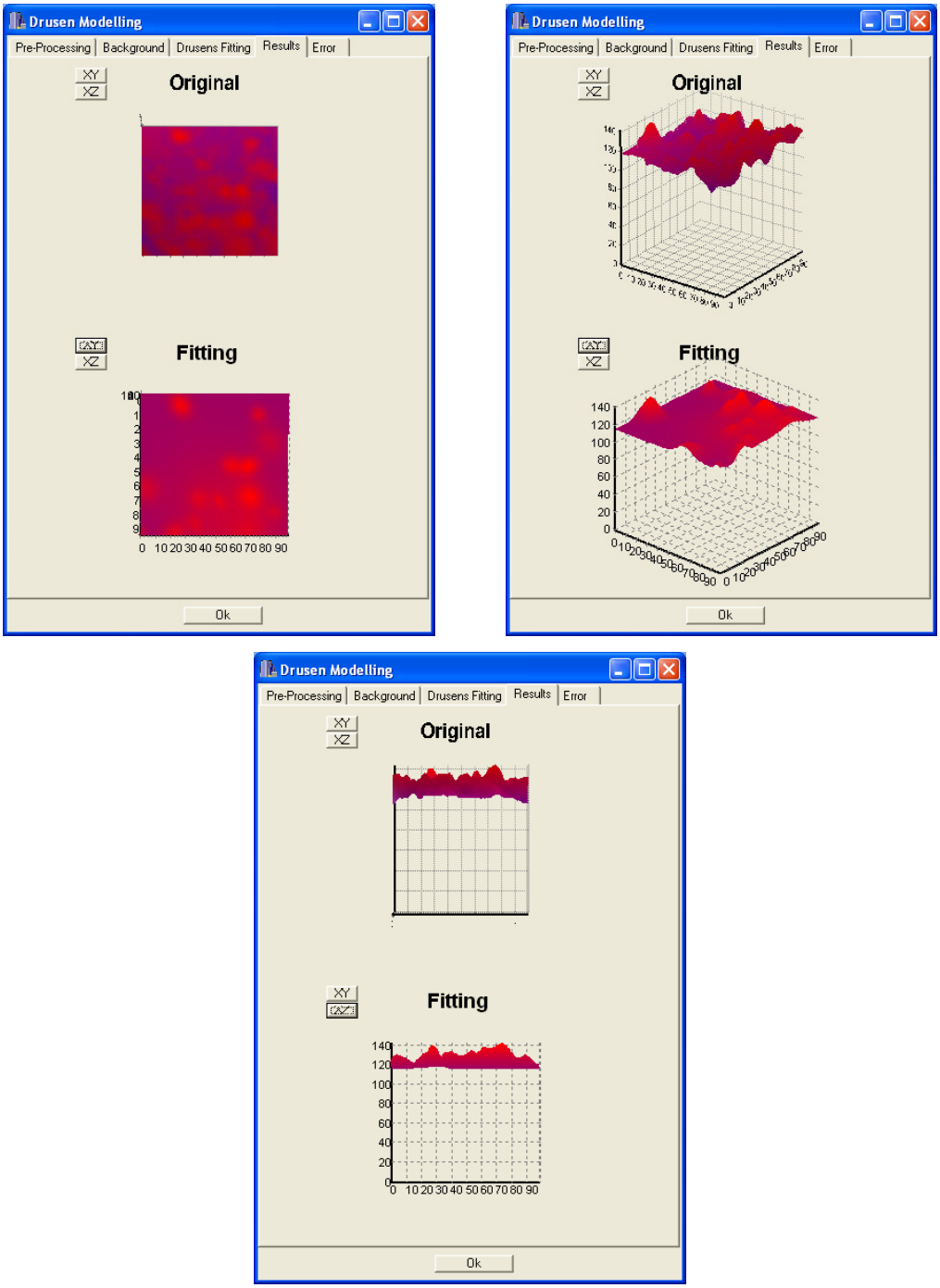


Figura 10.20 – Vista do Form Results

Error

Este menu mostra onde se encontram os erros, ou seja, o que não foi possível alcançar assim como os valores do Chi-Square final do background e das drusas.

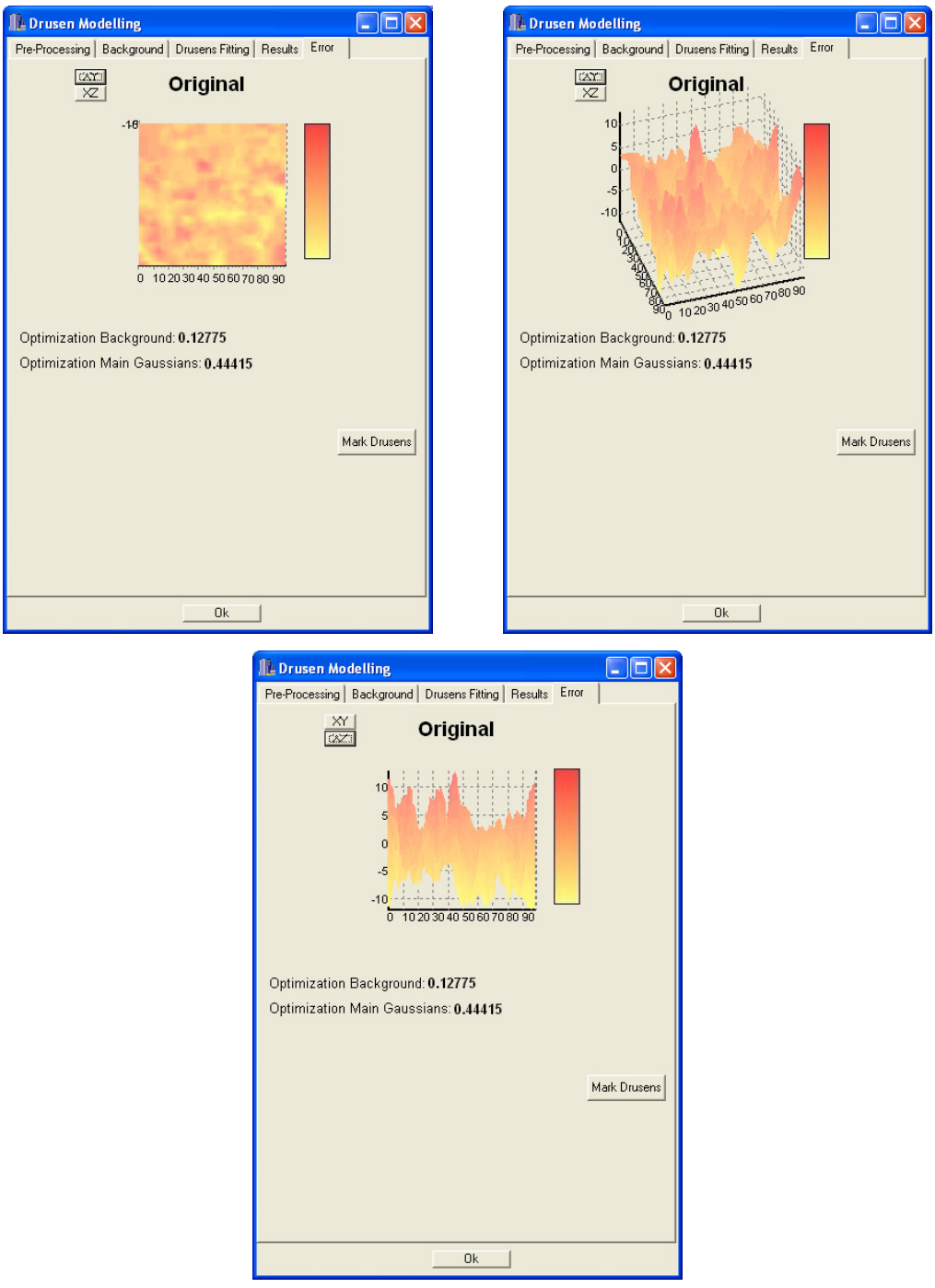


Figura 10.21 – Vista do Form Error

Oferece também a oportunidade de ver na imagem original o contorno das drusas após a modulação.

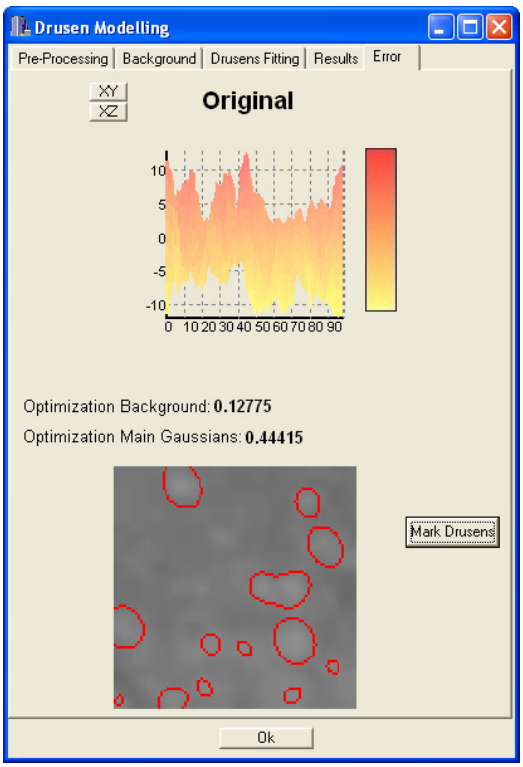


Figura 10.22 – Marcação do contorno das drusas

Anexo II - Listagem do Código

Gauss.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <stdio.h>
#include <math.h>
#include <iostream>
#include <iomanip>

#pragma hdrstop

#include "nr.h"
#include "Gauss.h"
#include "ChartInterface.h"

#pragma link "TeeSurfa"
#pragma link "TeePoin3"
#pragma link "TeeTools"
#pragma link "TeeDragPoint"
#pragma link "TeeOpenGL"
#pragma link "TeeTriSurface"
//#pragma resource "*.dfm"
//Gauss *GaussClass;

Gauss::Gauss()
{
        //inicializations
        NGaussian=1;
        Values.CountG=1;
        BackValues.CountG=0;
        AuxValues.CountG=1;
        //cont=0;
}

//destructor
Gauss::~Gauss(){

}

//---------------------------------------------------------------------------
int Gauss::LevMarq(float lin, float col,int BackOrGauss)
{

int ma;
if(BackOrGauss==1) ma=((9*((Values.CountG)-1)));
else ma=((9*((BackValues.CountG)-1)));

Vec_IO_DP x(col),y(lin),sig(col);
Vec_IO_DP a(ma);
Vec_IO_BOOL ia(ma);
Mat_O_DP covar(ma,ma),alpha(ma,ma);
DP chisq,ochisq,xi,alamda,yi;
Mat_DP z(col,lin);
float firstchisq;


  for(int i=0;i<Graph1.CountX;i++){
    x[i]=i;
  }
  for(int i=0;i<Graph1.CountY;i++){
    y[i]=i;
  }

  float kk=0;
  float Zmean=0;
  if(BackOrGauss==1){
    for(int i=0;i<Graph1.CountY;i++){
      for(int j=0;j<Graph1.CountX;j++){
        z[j][i]=Graph1.Val[j][i]; //i,j
        kk++;
        Zmean+=(1/kk)*(z[j][i]-Zmean);  //media recursiva
      }
    }
  }
  else{
    for(int i=0;i<Graph1.CountY;i++){
      for(int j=0;j<Graph1.CountX;j++){
        z[j][i]=Graph1.Val[j][i]; //i,j
        kk++;
        Zmean+=(1/kk)*(z[j][i]-Zmean);  //media recursiva
      }
    }
  }

  for(int i=0;i<col;i++){
    sig[i]=0.001;
  }

  int i,j;
  if(BackOrGauss==1){
    for (i=1,j=0;j<(9*((Values.CountG)-1));j+=9){
      a[j] = Values.GaussA[i].a;
      a[j+1] = Values.GaussA[i].y0;
      a[j+2] = Values.GaussA[i].x0;
      a[j+3] = Values.GaussA[i].Sx;
```

```
    a[j+4] = Values.GaussA[i].Sf;
    a[j+5] = Values.GaussA[i].d;
    a[j+6] = Values.GaussA[i].te;
    a[j+7] = Values.GaussA[i].Sy;//*Values.GaussA[j].Sf;
    a[j+8] = Values.GaussA[i].z0;

    ia[j] = true;
    ia[j+1] = true;
    ia[j+2] = true;
    ia[j+3] = true;
    ia[j+4] = false; //nao fazer
    ia[j+5] = false;
    ia[j+6] = false;
    ia[j+7] = true;
    ia[j+8] = false;

    i++;
  }

  a[9*(Values.CountG-1)-1]=Zmean;
  ia[9*(Values.CountG-1)-1]=true;
}
else{
 for (i=1,j=0;j<(9*((BackValues.CountG)-1));j+=9){
    a[j] = BackValues.GaussA[i].a-Zmean;
    a[j+1] = BackValues.GaussA[i].y0;
    a[j+2] = BackValues.GaussA[i].x0;
    a[j+3] = BackValues.GaussA[i].Sx;
    a[j+4] = BackValues.GaussA[i].Sf;
    a[j+5] = BackValues.GaussA[i].d;
    a[j+6] = BackValues.GaussA[i].te;
    a[j+7] = BackValues.GaussA[i].Sy;//*Values.GaussA[j].Sf;
    a[j+8] = BackValues.GaussA[i].z0;

    ia[j] = true;
    ia[j+1] = true;
    ia[j+2] = true;
    ia[j+3] = true;
    ia[j+4] = false; //nao fazer
    ia[j+5] = false;
    ia[j+6] = false;
    ia[j+7] = true;
    ia[j+8] = false;

    i++;
  }

  a[9*(BackValues.CountG-1)-1]=Zmean;
  ia[9*(BackValues.CountG-1)-1]=true;
}
int iter=0;
int k,itst;
for (iter=0;iter<1;iter++) {
  alamda = -1;
  ChartInterface->ProgressBar1->Position=10000;
  NR::mrqmin(x,y,z,sig,a,ia,covar,alpha,chisq,NR::fgauss,alamda);
  k=1;
  itst=0;
  firstchisq=chisq;
  for (;;) {
    k++;
    ochisq=chisq;
    NR::mrqmin(x,y,z,sig,a,ia,covar,alpha,chisq,NR::fgauss,alamda);
    if(BackOrGauss==1){
      ChartInterface->ProgressBar1->Position=((chisq/firstchisq)*10000);
      ChartInterface->Label3->Caption=FloatToStrF((chisq/firstchisq),0.0001,5,5);
    }
    else{
      ChartInterface->ProgressBar2->Position=((chisq/firstchisq)*10000);
      ChartInterface->Label2->Caption=FloatToStrF((chisq/firstchisq),0.0001,5,5);
    }
    iter++;
    fabs(ochisq-chisq) < 0.1 ? itst++ : itst=0;   //itst=0
    if((chisq/firstchisq)<0.12) itst=4;
    if (itst < 4) continue;
    alamda=0.0;
    //aux=chisq;
    aux=(chisq/firstchisq);

 //  NR::mrqmin(x,y,z,sig,a,ia,covar,alpha,chisq,NR::fgauss,alamda);
    break;
  }
}

ClearValues();
if(BackOrGauss==1){
  for(int j=0;j<ma;j+=9){
    Values.GaussA[Values.CountG].a = a[j];
    Values.GaussA[Values.CountG].x0 = a[j+1];
    Values.GaussA[Values.CountG].y0 = a[j+2];
    Values.GaussA[Values.CountG].Sx = a[j+3];
    Values.GaussA[Values.CountG].Sf = a[j+7]/a[j+3];
    Values.GaussA[Values.CountG].d = a[j+5];
    Values.GaussA[Values.CountG].te = a[j+6];
    Values.GaussA[Values.CountG].Sy = a[j+7];
    Values.GaussA[Values.CountG].z0 = a[j+8];
    Values.CountG++;
  }
```

```
 }
 else{
  for(int j=0;j<ma;j+=9){
    BackValues.GaussA[BackValues.CountG].a = a[j];
    BackValues.GaussA[BackValues.CountG].x0 = a[j+1];
    BackValues.GaussA[BackValues.CountG].y0 = a[j+2];
    BackValues.GaussA[BackValues.CountG].Sx = a[j+3];
    BackValues.GaussA[BackValues.CountG].Sf = a[j+7]/a[j+3];
    BackValues.GaussA[BackValues.CountG].d = a[j+5];
    BackValues.GaussA[BackValues.CountG].te = a[j+6];
    BackValues.GaussA[BackValues.CountG].Sy = a[j+7];
    BackValues.GaussA[BackValues.CountG].z0 = a[j+8];
    BackValues.CountG++;
  }
 }

//  return 1;

}

//---------------------------------------------------------------------------
//draw sum of Gaussian
int Gauss::GaussianCalc(float a,float x0, float y0, float Sx, float Sf, float d, float te, float z0, float
width, float height, int BackOrGauss)
{
TDateTime ti = ti.CurrentTime();

 double z=0;     float A,B;
 double Sy=Sx*Sf;
 cont++;
 float auxCos = cos(te);
 float auxSen = sin(te);
 float C=2/d;  int x,y;

 float xf,yf;
 for(x=0;x<width;x++){
      xf=x-x0;
      for(y=0;y<height;y++){
            yf=y-y0;
            A=(xf*auxCos+yf*auxSen)/Sx;
            B=(-xf*auxSen+yf*auxCos)/Sy;
            z=a*exp(-0.5*std::pow((A*A)+(B*B),C))+z0;
            if(BackOrGauss==1) Graph2.Val[13][y] +=z;
            if(BackOrGauss==2) auxGraph.Val[13][y]+=z;
            if(BackOrGauss==4) aux2Graph.Val[13][y]=z;

      }
 }

 TDateTime t1 = t1.CurrentTime();

 ChartInterface->tt += t1-ti;

 return 1;
}
//---------------------------------------------------------------------------
void Gauss::DrawAllValues(int height, int width,int BackOrGauss)
{
 if(BackOrGauss==1){
  for (int i=1;i<Values.CountG;i++){
   GaussianCalc(Values.GaussA[i].a,Values.GaussA[i].x0,
      Values.GaussA[i].y0,Values.GaussA[i].Sx,
      Values.GaussA[i].Sf,Values.GaussA[i].d,
      Values.GaussA[i].te,Values.GaussA[i].z0,width,height,1);
   ChartInterface->Label1->Caption=IntToStr(i);
  }
 }
 else{
  for (int i=1;i<BackValues.CountG;i++){
   GaussianCalc(BackValues.GaussA[i].a,BackValues.GaussA[i].x0,
     BackValues.GaussA[i].y0,BackValues.GaussA[i].Sx,
     BackValues.GaussA[i].Sf,BackValues.GaussA[i].d,
     BackValues.GaussA[i].te,BackValues.GaussA[i].z0,width,height,2);
     ChartInterface->Label1->Caption=IntToStr(i);
  }
 }
}


//---------------------------------------------------------------------------
void Gauss::Draw(int height, int width, int BackOrGauss)
{
 float erro=0;
 if(BackOrGauss==1){
  for(int i=0;i<height;i++){
   for(int j=0;j<width;j++){
    //erro+=(Graph1.Val[j][i]-Graph2.Val[j][i]);
    ChartInterface->Series2->AddXYZ(i,Graph2.Val[j][i],j);
    ChartInterface->Series3->AddXYZ(i,(Graph1.Val[j][i]-Graph2.Val[j][i]),j);
   }
  }
 }
 if(BackOrGauss==3){
  for(int i=0;i<height;i++){
   for(int j=0;j<width;j++){
    ChartInterface->SurfaceSeries1->AddXYZ(i,Graph2.Val[j][i]+auxGraph.Val[j][i],j);
    ChartInterface->SurfaceSeries5->AddXYZ(i,Graph1_copy.Val[j][i]-
(Graph2.Val[j][i]+auxGraph.Val[j][i]),j);
```

```
      }
    }
  }
  else{
    for(int i=0;i<height;i++){
      for(int j=0;j<width;j++){
        //erro+=(Graph1.Val[j][i]-Graph2.Val[j][i]);
        //ChartInterface->Series2->AddXYZ(i,Graph2.Val[j][i],j);
        ChartInterface->Series3->AddXYZ(i,auxGraph.Val[j][i],j);
      }
    }
  }

  erro=erro/(height*width);
  //ChartInterface->Label2->Caption=FloatToStr(erro);

}


//---------------------------------------------------------------------------
void Gauss::DrawSelectedValues(int width, int height, int last)
{
  while((AuxValues.CountG-1)!=0){
       GaussianCalc(AuxValues.GaussA[AuxValues.CountG-
1].a,AuxValues.GaussA[AuxValues.CountG-1].x0,
         AuxValues.GaussA[AuxValues.CountG-1].y0,AuxValues.GaussA[AuxValues.CountG-1].Sx,
         AuxValues.GaussA[AuxValues.CountG-1].Sf,AuxValues.GaussA[AuxValues.CountG-1].d,
         AuxValues.GaussA[AuxValues.CountG-1].te,AuxValues.GaussA[AuxValues.CountG-
1].z0,width,height,4);
         AuxValues.CountG--;
  }


GaussianCalc(AuxValues.GaussA[AuxValues.CountG].a,AuxValues.GaussA[AuxValues.CountG].
x0,
         AuxValues.GaussA[AuxValues.CountG].y0,AuxValues.GaussA[AuxValues.CountG].Sx,
         AuxValues.GaussA[AuxValues.CountG].Sf,AuxValues.GaussA[AuxValues.CountG].d,

AuxValues.GaussA[AuxValues.CountG].te,AuxValues.GaussA[AuxValues.CountG].z0,width,height
,4);
         //AuxValues.CountG--;

  ChartInterface->Series6->Clear();
  ChartInterface->Series6-
>AddXYZ(AuxValues.GaussA[AuxValues.CountG].y0,AuxValues.GaussA[AuxValues.CountG].a+A
uxValues.GaussA[AuxValues.CountG].z0,AuxValues.GaussA[AuxValues.CountG].x0);

  for(int i=0;i<height;i++){
     for(int j=0;j<width;j++){
        ChartInterface->Series5->AddXYZ(i,aux2Graph.Val[j][i],j);
      }
    }
}

//---------------------------------------------------------------------------

//draw sum of Gaussian
float Gauss::GaussianError()
{
  float error=0;
       ChartInterface->Series3->NumXValues = Graph1.CountX;
       ChartInterface->Series3->NumZValues = Graph1.CountY;

       for (int x=0;x<=Graph1.CountX;x++){
             for (int y=0;y<=Graph1.CountY;y++){
                   Graph3.Val[13][y]=Graph1.Val[13][y]-Graph4.Val[13][y];
                   error=error+(Graph3.Val[13][y]);
                   ChartInterface->Series3->AddXYZ(x,Graph3.Val[13][y],y);
       }
  }
  return error;
}


//---------------------------------------------------------------------------
void Gauss::Clears()
{
for(int x=0;x<=Graph4.CountX;x++){
       for(int y=0;y<=Graph4.CountY;y++){
             Graph4.Val[13][y]=0;
             Graph3.Val[13][y]=0;
             Graph2.Val[13][y]=0;
             ChartInterface->Series2->AddXYZ(y,0,x);
             ChartInterface->Series3->AddXYZ(y,0,x);
       }
  }
  ChartInterface->Series2->NumZValues =0;
  NGaussian=1;
}


//---------------------------------------------------------------------------
void Gauss::Reset()
{
  for(int x=0;x<=Graph4.CountX;x++){
    for(int y=0;y<=Graph4.CountY;y++){
      Graph4.Val[13][y]=0;
      Graph3.Val[13][y]=0;
```

```
      Graph2.Val[13][y]=0;
      auxGraph.Val[13][y]=0;
      ChartInterface->Series2->AddXYZ(y,0,x);
      ChartInterface->Series3->AddXYZ(y,0,x);
    }
  }
  ChartInterface->Series2->NumZValues=0;
  NGaussian=1;
  for(int i=0;i<500;i++){
    AuxValues.GaussA[i].a=0;
    AuxValues.GaussA[i].y0=0;
    AuxValues.GaussA[i].x0=0;
    AuxValues.GaussA[i].Sx=0;
    AuxValues.GaussA[i].Sf=0;
    AuxValues.GaussA[i].d=0;
    AuxValues.GaussA[i].te=0;
    AuxValues.GaussA[i].Sy=0;
    AuxValues.GaussA[i].z0=0;
  }
  Values.CountG=1;
}


//---------------------------------------------------------------------------
void Gauss::ClearValues()
{
  for(int i=0;i<Values.CountG;i++){
    Values.GaussA[i].a=0;
    Values.GaussA[i].x0=0;
    Values.GaussA[i].y0=0;
    Values.GaussA[i].Sx=0;
    Values.GaussA[i].Sf=0;
    Values.GaussA[i].d=0;
    Values.GaussA[i].te=0;
    Values.GaussA[i].z0=0;
  }

  for(int i=0;i<Values.CountG;i++){
    BackValues.GaussA[i].a=0;
    BackValues.GaussA[i].x0=0;
    BackValues.GaussA[i].y0=0;
    BackValues.GaussA[i].Sx=0;
    BackValues.GaussA[i].Sf=0;
    BackValues.GaussA[i].d=0;
    BackValues.GaussA[i].te=0;
    BackValues.GaussA[i].z0=0;
  }

  for(int i=1;i<AuxValues.CountG;i++){
    AuxValues.GaussA[i].a=0;
    AuxValues.GaussA[i].x0=0;
    AuxValues.GaussA[i].y0=0;
    AuxValues.GaussA[i].Sx=0;
    AuxValues.GaussA[i].Sf=0;
    AuxValues.GaussA[i].d=0;
    AuxValues.GaussA[i].te=0;
    AuxValues.GaussA[i].z0=0;
  }
  Values.CountG=1;
  BackValues.CountG=1;
  AuxValues.CountG=1;
}


//---------------------------------------------------------------------------
float Gauss::CalcError(int lin,int col)
{
  return Graph3.Val[lin][col]=Graph1.Val[lin][col]-Graph2.Val[lin][col];
}

//#pragma package(smart_init)
```

ChartInterface.Cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <stdio.h>
#pragma hdrstop
#include "Main.h"
#include "ChartInterface.h"
#include "Gauss.h"
//#include "nr.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "TeeSurfa"
#pragma link "TeeTools"
#pragma link "TeePoin3"
#pragma link "TeeDragPoint"
#pragma link "TeeOpenGL"
#pragma link "TeeTriSurface"
#pragma link "TeeOpenGL"
#pragma link "TeeSelectorTool"
#pragma link "TeePageNumTool"
#pragma resource "*.dfm"
TChartInterface *ChartInterface;

//---------------------------------------------------------------------------
__fastcall TChartInterface::TChartInterface(TComponent* Owner)
        : TForm(Owner)
{
  StringGrid1->DefaultColWidth = 36;
  StringGrid1->DefaultRowHeight = 15;
  StringGrid1->RowCount =2;
  StringGrid1->Cells[1][0]="a";
  StringGrid1->Cells[2][0]="x0";
  StringGrid1->Cells[3][0]="Sx";
  StringGrid1->Cells[4][0]="y0";
  StringGrid1->Cells[5][0]="Sf";
  StringGrid1->Cells[6][0]="Sy";
  StringGrid1->Cells[7][0]="d";
  StringGrid1->Cells[14][0]="te";
  StringGrid1->Cells[9][0]="z0";

  StringGrid2->DefaultColWidth = 36;
  StringGrid2->DefaultRowHeight = 15;
  StringGrid2->RowCount =2;
  StringGrid2->Cells[1][0]="a";
  StringGrid2->Cells[2][0]="x0";
  StringGrid2->Cells[3][0]="Sx";
  StringGrid2->Cells[4][0]="y0";
  StringGrid2->Cells[5][0]="Sf";
  StringGrid2->Cells[6][0]="Sy";
  StringGrid2->Cells[7][0]="d";
  StringGrid2->Cells[8][0]="te";
  StringGrid2->Cells[9][0]="z0";

  GaussClass = NULL;
 /* ImageClass->IfGrey();
  bool ecinza=currentWnd->ImageClass->IfGrey();
  if(ecinza)
    Button29->Enabled=false;*/

}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button1Click(TObject *Sender)
{
        ModalResult = mrOk;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button4Click(TObject *Sender)
{
 FILE *stream;
  char FirstLine[10000];
  char Num[1];
  String aux;

  if (GaussClass == NULL)
    GaussClass =  new Gauss();

  StringGrid2->RowCount=1;

  GaussClass->Values.CountG=1;

  int aux1=GaussClass->Values.CountG;

  int aux2=GaussClass->Values.CountG;
  GaussClass->NGaussian=GaussClass->Values.CountG;

  OpenDialog1->Options.Clear();
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute()){
```

```
    stream = fopen(OpenDialog1->FileName.c_str(), "r");
    if (stream){
        // read the first line from the file
        fgets(FirstLine, sizeof(FirstLine), stream);
        int j=0;
        int i=-1;
        int param=1;

        while(FirstLine[i]!='/'){
            i++;
            if((FirstLine[i]!=';')&&(FirstLine[i]!='/')){
                aux.Insert(FirstLine[i],aux.Length()+1);
            }
            if(param==9){
                param=1;
                aux1=++GaussClass->Values.CountG;
            }
            if (FirstLine[i]==';') {
              switch (param) {
               case 8:    GaussClass->Values.GaussA[aux1].z0=StrToFloat(aux);
                          aux2=++GaussClass->NGaussian;
                          StringGrid2->Cells[0][aux2-1] = IntToStr(aux2-1);
                          StringGrid2->Cells[1][aux2-1] = GaussClass->Values.GaussA[aux1].a;
                          StringGrid2->Cells[2][aux2-1] = GaussClass->Values.GaussA[aux1].x0;
                          StringGrid2->Cells[3][aux2-1] = GaussClass->Values.GaussA[aux1].Sx;
                          StringGrid2->Cells[4][aux2-1] = GaussClass->Values.GaussA[aux1].y0;
                          StringGrid2->Cells[5][aux2-1] = GaussClass->Values.GaussA[aux1].Sf;
                          StringGrid2->Cells[6][aux2-1]                = GaussClass-
>Values.GaussA[aux1].Sy=GaussClass->Values.GaussA[aux1].Sx*GaussClass-
>Values.GaussA[aux1].Sf;
                          StringGrid2->Cells[7][aux2-1] = GaussClass->Values.GaussA[aux1].d;
                          StringGrid2->Cells[8][aux2-1] = GaussClass->Values.GaussA[aux1].te;
                          StringGrid2->Cells[9][aux2-1] = GaussClass->Values.GaussA[aux1].z0;
                          StringGrid2->RowCount++;
                          break;
               case 7:    GaussClass->Values.GaussA[aux1].te=StrToFloat(aux);
                          break;
               case 6:    GaussClass->Values.GaussA[aux1].d=StrToFloat(aux);
                          break;
               case 5:    GaussClass->Values.GaussA[aux1].Sf=StrToFloat(aux);
                          break;
               case 4:    GaussClass->Values.GaussA[aux1].y0=StrToFloat(aux);
                          break;
               case 3:    GaussClass->Values.GaussA[aux1].Sx=StrToFloat(aux);
                          break;
               case 2:    GaussClass->Values.GaussA[aux1].x0=StrToFloat(aux);
                          break;
               case 1:    GaussClass->Values.GaussA[aux1].a=StrToFloat(aux);
                          break;
                }
                param++;
                aux.Delete(1,10);
              }
            }
            fclose(stream);
        }

        GaussClass->Values.GaussA[aux1].z0=StrToFloat(aux);
        aux1=GaussClass->Values.CountG++;
        aux2=++GaussClass->NGaussian;

        StringGrid2->Cells[0][aux2-1] = IntToStr(aux2-1);
        StringGrid2->Cells[1][aux2-1] = GaussClass->Values.GaussA[aux1].a;
        StringGrid2->Cells[2][aux2-1] = GaussClass->Values.GaussA[aux1].x0;
        StringGrid2->Cells[3][aux2-1] = GaussClass->Values.GaussA[aux1].Sx;
        StringGrid2->Cells[4][aux2-1] = GaussClass->Values.GaussA[aux1].y0;
        StringGrid2->Cells[5][aux2-1] = GaussClass->Values.GaussA[aux1].Sf;
        StringGrid2->Cells[6][aux2-1]      =      GaussClass->Values.GaussA[aux1].Sy=GaussClass-
>Values.GaussA[aux1].Sx*GaussClass->Values.GaussA[aux1].Sf;
        StringGrid2->Cells[7][aux2-1] = GaussClass->Values.GaussA[aux1].d;
        StringGrid2->Cells[8][aux2-1] = GaussClass->Values.GaussA[aux1].te;
        StringGrid2->Cells[9][aux2-1] = GaussClass->Values.GaussA[aux1].z0;
        StringGrid2->RowCount++;
    }
    //unsigned short hour; unsigned short min; unsigned short sec; unsigned short msec;
    //tt.DecodeTime(&hour, &min, &sec, &msec);
    //msec += sec*1000;
    //Application->MessageBox("Time", (IntToStr(msec).c_str()), MB_OK);
}


//---------------------------------------------------------------------------
//comute to edit cell
void __fastcall TChartInterface::RadioButton1Click(TObject *Sender)
{
  StringGrid2->Options.operator <<(goEditing);
  StringGrid2->Options.operator >>(goRowSelect);
}


//---------------------------------------------------------------------------
//comute to select row
void __fastcall TChartInterface::RadioButton2Click(TObject *Sender)
{
  StringGrid2->Options.operator >>(goEditing);
  StringGrid2->Options.operator <<(goRowSelect);
}
```

```
//---------------------------------------------------------------------------
void __fastcall TChartInterface::StringGrid2SelectCell(TObject *Sender, int ACol,
      int ARow, bool &CanSelect)
{
  int line;
  if(RadioButton2->Checked){
    line=ARow;
    SelectedRow2=line;
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].z0=StrToFloat(StringGrid2-
>Cells[9][StringGrid2->RowCount-2]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].te=StrToFloat(StringGrid2-
>Cells[8][15]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].d=StrToFloat(StringGrid2-
>Cells[7][15]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].Sf=StrToFloat(StringGrid2-
>Cells[5][15]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].y0=StrToFloat(StringGrid2-
>Cells[4][15]);
    GaussClass->AuxValues.GaussA[GaussClass-
>AuxValues.CountG].Sx=StrToFloat(StringGrid2->Cells[3][15]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].x0=StrToFloat(StringGrid2-
>Cells[2][15]);
    GaussClass->AuxValues.GaussA[GaussClass->AuxValues.CountG].a=StrToFloat(StringGrid2-
>Cells[1][15]);
    GaussClass->AuxValues.CountG++;

    GaussClass->DrawSelectedValues(GaussClass->img_height,GaussClass-
>img_width,StringGrid2->RowCount);
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button5Click(TObject *Sender)
{
  TCursor Save_Cursor = Screen->Cursor;
            Screen->Cursor = crHourGlass;    // Show hourglass cursor

  int height=currentWnd->ImageClass->getHeight();
  int width=currentWnd->ImageClass->getWidth();
  Edit16->Text=IntToStr(height);
  Edit17->Text=IntToStr(width);

  for(int i=0;i<height;i++){
    for (int j=0;j<width;j++){
      ChartInterface->Series1->AddXYZ(i,0,j);
      ChartInterface->Series2->AddXYZ(i,0,j);
      ChartInterface->Series3->AddXYZ(i,0,j);
      ChartInterface->SurfaceSeries3->AddXYZ(i,0,j);
    }
  }

  if  (GaussClass == NULL)
    GaussClass =  new Gauss();

  GaussClass->img_height=height;
  GaussClass->img_width=width;
  int z=0;
  GaussClass->Graph1.CountX = width;        //j
  GaussClass->Graph1.CountY = height;       //i
  GaussClass->Graph1_copy.CountX = width;        //j
  GaussClass->Graph1_copy.CountY = height;       //i
  for(int i=0;i<height;i++){
    for (int j=0;j<width;j++){
      z = currentWnd->ImageClass->getColorValue(j,i);
      Series1->AddXYZ(i,z,j);
      ChartInterface->SurfaceSeries3->AddXYZ(i,z,j);
      // colocar para uma struct minha
      GaussClass->Graph1.Val[j][i]=z;
      GaussClass->Graph1_copy.Val[j][i]=z;
    }
  }

  TeeOpenGL1->TeePanel = Chart1;
  TeeOpenGL1->TeePanel = Chart2;
  TeeOpenGL1->TeePanel = Chart3;
  TeeOpenGL1->TeePanel = Chart4;
  TeeOpenGL1->TeePanel = Chart5;
  TeeOpenGL1->Active = True;

  Screen->Cursor = Save_Cursor; // always restore the cursor

}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button2Click(TObject *Sender)
{
  Series2->Clear();
  GaussClass->DrawAllValues(GaussClass->img_height,GaussClass->img_width,1);
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,1);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button7Click(TObject *Sender)
{
```

```
  for(int i= SelectedRow2;i<StringGrid2->RowCount-2;i++){
    for(int j=0;j<StringGrid2->ColCount-1;j++)
      StringGrid2->Cells[j][i]=StringGrid2->Cells[j][i+1];
  }
  for(int i=1;i<StringGrid2->RowCount-2;i++)
    StringGrid2->Cells[0][i]=i;

  StringGrid2->RowCount--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button8Click(TObject *Sender)
{
  StringGrid2->RowCount++;
  GaussClass->Values.CountG++;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button6Click(TObject *Sender)
{
  for(int i= SelectedRow1;i<StringGrid1->RowCount-2;i++){
    for(int j=0;j<StringGrid1->ColCount-1;j++)
      StringGrid1->Cells[j][i]=StringGrid1->Cells[j][i+1];
  }
  for(int i=1;i<StringGrid1->RowCount-2;i++)
    StringGrid1->Cells[0][i]=i;

  StringGrid1->RowCount--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::FormShow(TObject *Sender)
{
  Image1->Picture=currentWnd->ImageApl->Picture;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button10Click(TObject *Sender)
{
  StringGrid2->RowCount=2;
  GaussClass->ClearValues();
  int aux = currentWnd->ImageClass->seqWatershed->maxIndex;
  int auxCont=1;
  for(int i=0;i<aux;i++){
    if(currentWnd->ImageClass->seqWatershed->maxInfo[i].valid){
      StringGrid2->Cells[0][auxCont]=IntToStr(auxCont);
      StringGrid2->Cells[1][auxCont]=IntToStr(currentWnd->ImageClass->seqWatershed-
>maxInfo[i].maxAmplitude);
      StringGrid2->Cells[2][auxCont]=IntToStr(currentWnd->ImageClass->seqWatershed-
>maxInfo[i].lin);
      StringGrid2->Cells[3][auxCont]=IntToStr(2);
      StringGrid2->Cells[4][auxCont]=IntToStr(currentWnd->ImageClass->seqWatershed-
>maxInfo[i].col);
      StringGrid2->Cells[5][auxCont]=IntToStr(1);
      StringGrid2->Cells[6][auxCont]=IntToStr(2);
      StringGrid2->Cells[7][auxCont]=IntToStr(2);
      StringGrid2->Cells[8][auxCont]=IntToStr(1);
      StringGrid2->Cells[9][auxCont]=IntToStr(0);

      GaussClass->Values.GaussA[auxCont].a=currentWnd->ImageClass->seqWatershed-
>maxInfo[i].maxAmplitude;
      GaussClass->Values.GaussA[auxCont].x0=currentWnd->ImageClass->seqWatershed-
>maxInfo[i].lin;
      GaussClass->Values.GaussA[auxCont].Sx=2;
      GaussClass->Values.GaussA[auxCont].y0=currentWnd->ImageClass->seqWatershed-
>maxInfo[i].col;
      GaussClass->Values.GaussA[auxCont].Sf=1;
      GaussClass->Values.GaussA[auxCont].Sy=2;
      GaussClass->Values.GaussA[auxCont].d=2;
      GaussClass->Values.GaussA[auxCont].te=0;
      GaussClass->Values.GaussA[auxCont].z0=0;

      StringGrid2->RowCount++;
      auxCont++;
      GaussClass->Values.CountG=auxCont;
    }
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button11Click(TObject *Sender)
{
  TCursor Save_Cursor = Screen->Cursor;
            Screen->Cursor = crHourGlass;    // Show hourglass cursor

  currentWnd->ImageClass->seqWatershed = new TSeqWatershed(currentWnd->ImageClass-
>pictureCopy);
  Label1->Caption=SeqWatershedF->numberObj->Text;

  Screen->Cursor = Save_Cursor; // always restore the cursor
}
```

```
//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button12Click(TObject *Sender)
{
TCursor Save_Cursor = Screen->Cursor;
        Screen->Cursor = crHourGlass;    // Show hourglass cursor

  Button14Click(Sender);
  GaussClass->LevMarq(GaussClass->img_height,GaussClass->img_width,1);
  int auxCont = GaussClass->Values.CountG;

  for (int i=1;i<auxCont;i++){
    StringGrid2->Cells[0][i]=IntToStr(i);
    StringGrid2->Cells[1][i]=FloatToStr(GaussClass->Values.GaussA[i].a);
    StringGrid2->Cells[2][i]=FloatToStr(GaussClass->Values.GaussA[i].x0);
    StringGrid2->Cells[3][i]=FloatToStr(GaussClass->Values.GaussA[i].Sx);
    StringGrid2->Cells[4][i]=FloatToStr(GaussClass->Values.GaussA[i].y0);
    StringGrid2->Cells[5][i]=FloatToStr(GaussClass->Values.GaussA[i].Sf);
    StringGrid2->Cells[6][i]=FloatToStr(GaussClass->Values.GaussA[i].Sy);
    StringGrid2->Cells[7][i]=FloatToStr(GaussClass->Values.GaussA[i].d);
    StringGrid2->Cells[8][i]=FloatToStr(GaussClass->Values.GaussA[i].te);
    StringGrid2->Cells[9][i]=FloatToStr(GaussClass->Values.GaussA[i].z0);
  }
  Label3->Caption=FloatToStrF(GaussClass->aux,0.0001,5,5);
  Label10->Caption=FloatToStrF(GaussClass->aux,0.0001,5,5);
  GaussClass->DrawAllValues(GaussClass->img_height,GaussClass->img_width,1);
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,1);
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,3);
  Screen->Cursor = Save_Cursor; // always restore the cursor
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button13Click(TObject *Sender)
{
/*  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,1);
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,3);*/
StringGrid1->RowCount =4;
  StringGrid1->Cells[0][1]=1;
  StringGrid1->Cells[1][1]=120;
  StringGrid1->Cells[2][1]=GaussClass->img_height/2;
  StringGrid1->Cells[3][1]=GaussClass->img_height;
  StringGrid1->Cells[4][1]=GaussClass->img_width/2;
  StringGrid1->Cells[5][1]=1;
  StringGrid1->Cells[6][1]=GaussClass->img_height;
  StringGrid1->Cells[7][1]=2;
  StringGrid1->Cells[8][1]=1;
  StringGrid1->Cells[9][1]=0;
  StringGrid1->Cells[0][2]=2;
  StringGrid1->Cells[1][2]=50;
  StringGrid1->Cells[2][2]=GaussClass->img_height/2;
  StringGrid1->Cells[3][2]=GaussClass->img_height/2;
  StringGrid1->Cells[4][2]=GaussClass->img_width/2;
  StringGrid1->Cells[5][2]=1;
  StringGrid1->Cells[6][2]=GaussClass->img_height/2;
  StringGrid1->Cells[7][2]=2;
  StringGrid1->Cells[8][2]=1;
  StringGrid1->Cells[9][2]=0;

  StringGrid1->Cells[0][3]=3;
  StringGrid1->Cells[1][3]=-20;
  StringGrid1->Cells[2][3]=GaussClass->img_height/2;
  StringGrid1->Cells[3][3]=GaussClass->img_height/3;
  StringGrid1->Cells[4][3]=GaussClass->img_width/2;
  StringGrid1->Cells[5][3]=1;
  StringGrid1->Cells[6][3]=GaussClass->img_height/3;
  StringGrid1->Cells[7][3]=2;
  StringGrid1->Cells[8][3]=1;
  StringGrid1->Cells[9][3]=0;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button14Click(TObject *Sender)
{
  GaussClass->Values.CountG=1;
  for(int i=1;i<StringGrid2->RowCount-1;i++){
    GaussClass->Values.GaussA[i].a=StrToFloat(StringGrid2->Cells[1][i]);
    GaussClass->Values.GaussA[i].x0=StrToFloat(StringGrid2->Cells[2][i]);
    GaussClass->Values.GaussA[i].Sx=StrToFloat(StringGrid2->Cells[3][i]);
    GaussClass->Values.GaussA[i].y0=StrToFloat(StringGrid2->Cells[4][i]);
    GaussClass->Values.GaussA[i].Sf=StrToFloat(StringGrid2->Cells[5][i]);
    GaussClass->Values.GaussA[i].Sy=StrToFloat(StringGrid2->Cells[6][i]);
    GaussClass->Values.GaussA[i].d=StrToFloat(StringGrid2->Cells[7][i]);
    GaussClass->Values.GaussA[i].te=StrToFloat(StringGrid2->Cells[8][i]);
    GaussClass->Values.GaussA[i].z0=StrToFloat(StringGrid2->Cells[9][i]);
    GaussClass->Values.CountG++;
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button16Click(TObject *Sender)
{
 FILE *stream;
  char FirstLine[10000];
  char Num[1];
```

```
  char aux[10000];
  int cont=0;
  int i;
  AnsiString ansiaux;

  int aux1=StringGrid2->RowCount-1;

  int aux2=GaussClass->Values.CountG;

  OpenDialog1->Options.Clear();
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute()){
       int numero=0;
       stream = fopen(OpenDialog1->FileName.c_str(), "w");
       for(int j=1;j<StringGrid2->RowCount-1;j++){
         for(int w=1;w<10;w++){
           ansiaux=StringGrid2->Cells[w][j];
           numero=ansiaux.Length();
           if(w!=6){
             if(numero>8) numero=8;
             for(int h=1;h<numero+1;h++){
               aux[cont]=ansiaux[h];
               cont++;
             }
             aux[cont]=';';
             cont++;
           }
         }
       }
       aux[cont-1]='/';
       fwrite(&aux,cont,1,stream);
       fclose(stream);
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button17Click(TObject *Sender)
{
  Chart2->View3DOptions->Elevation=270;
  Chart2->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button18Click(TObject *Sender)
{
  GaussClass->Reset();
  for(int i=1;i<StringGrid2->RowCount;i++){
    StringGrid2->Cells[0][i]=0;
    StringGrid2->Cells[1][i]=0;
    StringGrid2->Cells[2][i]=0;
    StringGrid2->Cells[3][i]=0;
    StringGrid2->Cells[4][i]=0;
    StringGrid2->Cells[5][i]=0;
    StringGrid2->Cells[6][i]=0;
    StringGrid2->Cells[7][i]=0;
    StringGrid2->Cells[8][i]=0;
    StringGrid2->Cells[9][i]=0;
  }
  StringGrid2->RowCount=2;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button20Click(TObject *Sender)
{
  GaussClass->BackValues.CountG=1;
  for(int i=1;i<StringGrid1->RowCount;i++){
    GaussClass->BackValues.GaussA[i].a=StrToFloat(StringGrid1->Cells[1][i]);
    GaussClass->BackValues.GaussA[i].x0=StrToFloat(StringGrid1->Cells[2][i]);
    GaussClass->BackValues.GaussA[i].Sx=StrToFloat(StringGrid1->Cells[3][i]);
    GaussClass->BackValues.GaussA[i].y0=StrToFloat(StringGrid1->Cells[4][i]);
    GaussClass->BackValues.GaussA[i].Sf=StrToFloat(StringGrid1->Cells[5][i]);
    GaussClass->BackValues.GaussA[i].Sy=StrToFloat(StringGrid1->Cells[6][i]);
    GaussClass->BackValues.GaussA[i].d=StrToFloat(StringGrid1->Cells[7][i]);
    GaussClass->BackValues.GaussA[i].te=StrToFloat(StringGrid1->Cells[8][i]);
    GaussClass->BackValues.GaussA[i].z0=StrToFloat(StringGrid1->Cells[9][i]);
    GaussClass->BackValues.CountG++;
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button19Click(TObject *Sender)
{
  TCursor Save_Cursor = Screen->Cursor;
            Screen->Cursor = crHourGlass;    // Show hourglass cursor
  Button20Click(Sender);
  GaussClass->LevMarq(GaussClass->img_height,GaussClass->img_width,2);
  int auxCont = GaussClass->BackValues.CountG;

  for (int i=1;i<auxCont;i++){
    StringGrid1->Cells[0][i]=IntToStr(i);
    StringGrid1->Cells[1][i]=FloatToStr(GaussClass->BackValues.GaussA[i].a);
    StringGrid1->Cells[2][i]=FloatToStr(GaussClass->BackValues.GaussA[i].x0);
    StringGrid1->Cells[3][i]=FloatToStr(GaussClass->BackValues.GaussA[i].Sx);
```

```
    StringGrid1->Cells[4][i]=FloatToStr(GaussClass->BackValues.GaussA[i].y0);
    StringGrid1->Cells[5][i]=FloatToStr(GaussClass->BackValues.GaussA[i].Sf);
    StringGrid1->Cells[6][i]=FloatToStr(GaussClass->BackValues.GaussA[i].Sy);
    StringGrid1->Cells[7][i]=FloatToStr(GaussClass->BackValues.GaussA[i].d);
    StringGrid1->Cells[8][i]=FloatToStr(GaussClass->BackValues.GaussA[i].te);
    StringGrid1->Cells[9][i]=FloatToStr(GaussClass->BackValues.GaussA[i].z0);
  }
  Label2->Caption=FloatToStrF(GaussClass->aux,0.0001,5,5);
  Label9->Caption=FloatToStrF(GaussClass->aux,0.0001,5,5);
              Screen->Cursor = Save_Cursor; // always restore the cursor
  Application->MessageBox("If you wish you can subtract the Background to the image or you can
leave it that way", MB_OK);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button23Click(TObject *Sender)
{
  StringGrid1->RowCount++;
  GaussClass->BackValues.CountG++;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button21Click(TObject *Sender)
{
  GaussClass->DrawAllValues(GaussClass->img_height,GaussClass->img_width,2);
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,2);

  GaussClass->Graph1.CountX = currentWnd->ImageClass->getWidth();       //j
  GaussClass->Graph1.CountY = currentWnd->ImageClass->getHeight();      //i
  for(int i=0;i<GaussClass->Graph1.CountY;i++){
    //line = (byte *) currentWnd->ImageClass->pictureCopy->ScanLine[i];
    for (int j=0;j<GaussClass->Graph1.CountX;j++){
      GaussClass->Graph1.Val[j][i]=GaussClass->Graph1.Val[j][i]-GaussClass->auxGraph.Val[j][i];
      Series1->AddXYZ(i,GaussClass->Graph1.Val[j][i],j);
      //line[j*3] = GaussClass->Graph1.Val[j][i];
      //line[j*3+1] = GaussClass->Graph1.Val[j][i];
                                  //line[j*3+2] = GaussClass->Graph1.Val[j][i];
    }
  }
              //currentWnd->ImageClass->imgComp->Picture->Bitmap->Assign(currentWnd-
>ImageClass->pictureCopy);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button22Click(TObject *Sender)
{
  GaussClass->Draw(GaussClass->img_height,GaussClass->img_width,3);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button24Click(TObject *Sender)
{
 FILE *stream;
  char FirstLine[10000];
  char Num[1];
  String aux;

  if (GaussClass == NULL)
    GaussClass =  new Gauss();

  int aux1=GaussClass->BackValues.CountG;

  int aux2=GaussClass->BackValues.CountG;
  GaussClass->NGaussian=GaussClass->BackValues.CountG;

  OpenDialog1->Options.Clear();
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute()){
       stream = fopen(OpenDialog1->FileName.c_str(), "r");
       if (stream){
             // read the first line from the file
             fgets(FirstLine, sizeof(FirstLine), stream);
             int j=0;
             int i=-1;
             int param=1;

             while(FirstLine[i]!='/'){
                   i++;
                   if((FirstLine[i]!=';')&&(FirstLine[i]!='/')){
                         aux.Insert(FirstLine[i],aux.Length()+1);
                   }
                   if(param==9){
                         param=1;
                         aux1=++GaussClass->BackValues.CountG;
                   }
                   if (FirstLine[i]==';') {
                     switch (param) {
                      case 8:      GaussClass->BackValues.GaussA[aux1].z0=StrToFloat(aux);
                                   aux2=++GaussClass->NGaussian;
                                   StringGrid1->Cells[0][aux2] = IntToStr(aux2-1);
                                   StringGrid1->Cells[1][aux2] = GaussClass->BackValues.GaussA[aux1].a;
                                   StringGrid1->Cells[2][aux2]                =                GaussClass-
>BackValues.GaussA[aux1].x0;
```

```
                        StringGrid1->Cells[3][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].Sx;
                        StringGrid1->Cells[4][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].y0;
                        StringGrid1->Cells[5][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].Sf;
                        StringGrid1->Cells[6][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].Sy=GaussClass->BackValues.GaussA[aux1].Sx*GaussClass-
>BackValues.GaussA[aux1].Sf;
                        StringGrid1->Cells[7][aux2] = GaussClass->BackValues.GaussA[aux1].d;
                        StringGrid1->Cells[8][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].te;
                        StringGrid1->Cells[9][aux2]          =          GaussClass-
>BackValues.GaussA[aux1].z0;
                        StringGrid1->RowCount++;
                        break;
                 case 7:    GaussClass->BackValues.GaussA[aux1].te=StrToFloat(aux);
                        break;
                 case 6:    GaussClass->BackValues.GaussA[aux1].d=StrToFloat(aux);
                        break;
                 case 5:    GaussClass->BackValues.GaussA[aux1].Sf=StrToFloat(aux);
                        break;
                 case 4:    GaussClass->BackValues.GaussA[aux1].y0=StrToFloat(aux);
                        break;
                 case 3:    GaussClass->BackValues.GaussA[aux1].Sx=StrToFloat(aux);
                        break;
                 case 2:    GaussClass->BackValues.GaussA[aux1].x0=StrToFloat(aux);
                        break;
                 case 1:    GaussClass->BackValues.GaussA[aux1].a=StrToFloat(aux);
                        break;
                }
                param++;
                aux.Delete(1,10);
               }
          }
          fclose(stream);
     }

     GaussClass->BackValues.GaussA[aux1].z0=StrToFloat(aux);
     aux1=GaussClass->BackValues.CountG++;
     aux2=++GaussClass->NGaussian;

     StringGrid1->Cells[0][aux2] = IntToStr(aux2-1);
     StringGrid1->Cells[1][aux2] = GaussClass->BackValues.GaussA[aux1].a;
     StringGrid1->Cells[2][aux2] = GaussClass->BackValues.GaussA[aux1].x0;
     StringGrid1->Cells[3][aux2] = GaussClass->BackValues.GaussA[aux1].Sx;
     StringGrid1->Cells[4][aux2] = GaussClass->BackValues.GaussA[aux1].y0;
     StringGrid1->Cells[5][aux2] = GaussClass->BackValues.GaussA[aux1].Sf;
     StringGrid1->Cells[6][aux2]    =    GaussClass->BackValues.GaussA[aux1].Sy=GaussClass-
>BackValues.GaussA[aux1].Sx*GaussClass->BackValues.GaussA[aux1].Sf;
     StringGrid1->Cells[7][aux2] = GaussClass->BackValues.GaussA[aux1].d;
     StringGrid1->Cells[8][aux2] = GaussClass->BackValues.GaussA[aux1].te;
     StringGrid1->Cells[9][aux2] = GaussClass->BackValues.GaussA[aux1].z0;
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button3Click(TObject *Sender)
{
  FILE *stream;
  char FirstLine[10000];
  char Num[1];
  char aux[10000];
  int cont=0;
  int i;
  AnsiString ansiaux;

  int aux1=StringGrid1->RowCount;

  int aux2=GaussClass->BackValues.CountG;


  OpenDialog1->Options.Clear();
  OpenDialog1->Filter = "Text files (*.txt)|*.txt|All files (*.*)|*.*";
  OpenDialog1->FilterIndex = 2; // start the dialog showing all files
  if (OpenDialog1->Execute()){
       int numero=0;
       stream = fopen(OpenDialog1->FileName.c_str(), "w");
       for(int j=1;j<StringGrid1->RowCount;j++){
         for(int w=1;w<10;w++){
           ansiaux=StringGrid1->Cells[w][j];
           numero=ansiaux.Length();
           if(w!=6){
             if(numero>8) numero=8;
             for(int h=1;h<numero+1;h++){
               aux[cont]=ansiaux[h];
               cont++;
             }
             aux[cont]=';';
             cont++;
           }
         }
       }
       aux[cont-1]='/';
       fwrite(&aux,cont,1,stream);
       fclose(stream);
```

```
  }
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button9Click(TObject *Sender)
{
  Chart1->View3DOptions->Elevation=270;
  Chart1->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button15Click(TObject *Sender)
{
  Chart3->View3DOptions->Elevation=270;
  Chart3->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button25Click(TObject *Sender)
{
  Chart1->View3DOptions->Elevation=0;
  Chart1->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button26Click(TObject *Sender)
{
  Chart3->View3DOptions->Elevation=0;
  Chart3->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button27Click(TObject *Sender)
{
  Chart2->View3DOptions->Elevation=0;
  Chart2->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button28Click(TObject *Sender)
{
  TCursor Save_Cursor = Screen->Cursor;
        Screen->Cursor = crHourGlass;    // Show hourglass cursor

  currentWnd->ImageClass->AplyMatrix3(matrixF->matrix3,matrixF->weight);
  Image1->Picture=currentWnd->ImageApl->Picture;

        Screen->Cursor = Save_Cursor; // always restore the cursor


}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button29Click(TObject *Sender)
{
  TCursor Save_Cursor = Screen->Cursor;
        Screen->Cursor = crHourGlass;    // Show hourglass cursor

  currentWnd->ImageClass->convertToGrey();
  Image1->Picture=currentWnd->ImageApl->Picture;

        Screen->Cursor = Save_Cursor; // always restore the cursor
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button32Click(TObject *Sender)
{
  Chart5->View3DOptions->Elevation=270;
  Chart5->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button33Click(TObject *Sender)
{
  Chart5->View3DOptions->Elevation=0;
  Chart5->View3DOptions->Rotation=270;
}

//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button31Click(TObject *Sender)
{
  Chart4->View3DOptions->Elevation=0;
  Chart4->View3DOptions->Rotation=270;
}

//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button30Click(TObject *Sender)
{
  Chart4->View3DOptions->Elevation=270;
```

```
  Chart4->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button34Click(TObject *Sender)
{
  Chart6->View3DOptions->Elevation=270;
  Chart6->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button35Click(TObject *Sender)
{
  Chart6->View3DOptions->Elevation=0;
  Chart6->View3DOptions->Rotation=270;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Button36Click(TObject *Sender)
{
  int height=GaussClass->img_height;      //i
  int width=GaussClass->img_width;       //j
  float max=-30;
  float z_last=0;
  for(int i=0;i<height;i++){
    for (int j=0;j<width;j++){
      if (GaussClass->Graph2.Val[j][i]>max)
        max=GaussClass->Graph2.Val[j][i];
    }
  }
  z_last=GaussClass->Values.GaussA[GaussClass->Values.CountG-1].z0;
  max=((abs(z_last)+max)/4)+z_last;

  for(int i=0;i<height;i++){
    for (int j=0;j<width;j++){
      if(GaussClass->Graph2.Val[j][i]>max){
        if((GaussClass->Graph2.Val[j+1][i]<max)||
          (GaussClass->Graph2.Val[j][i-1]<max)||
          (GaussClass->Graph2.Val[j-1][i]<max)||
          (GaussClass->Graph2.Val[j][i+1]<max))
            currentWnd->ImageClass-> setColorValue(j,i);
      }
    }
  }
  Image2->Picture=currentWnd->ImageApl->Picture;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart2MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart2MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom++;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart1MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart1MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom++;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart3MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart3MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom++;
}
```

```
//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart5MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart5MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart4MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart4MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom++;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart6MouseWheelDown(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom--;
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::Chart6MouseWheelUp(TObject *Sender,
      TShiftState Shift, TPoint &MousePos, bool &Handled)
{
  Chart2->View3DOptions->Zoom++;
}


//---------------------------------------------------------------------------
 __fastcall TChartInterface::~TChartInterface()
{
  //TODO: Add your source code here

  delete GaussClass;

}

void __fastcall TChartInterface::RadioButton3Click(TObject *Sender)
{
  StringGrid1->Options.operator <<(goEditing);
  StringGrid1->Options.operator >>(goRowSelect);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::RadioButton4Click(TObject *Sender)
{
  StringGrid1->Options.operator >>(goEditing);
  StringGrid1->Options.operator <<(goRowSelect);
}


//---------------------------------------------------------------------------
void __fastcall TChartInterface::StringGrid1SelectCell(TObject *Sender,
      int ACol, int ARow, bool &CanSelect)
{
  SelectedRow1 = ARow;
}


//---------------------------------------------------------------------------
```

MrqMin.cpp

```
#include "nr.h"
#include <vcl\Forms.hpp>

void NR::mrqmin(Vec_I_DP &x, Vec_I_DP &y, Mat_I_DP &z, Vec_I_DP &sig, Vec_IO_DP &a,
		Vec_I_BOOL &ia, Mat_O_DP &covar, Mat_O_DP &alpha, DP &chisq,
		void funcs(const DP, const DP, Vec_I_DP &, DP &, Vec_O_DP &), DP &alamda)
{
		static int mfit;
		static DP ochisq;

		int j,k,l;
  bool valGaussJ;

		int ma=a.size();

  static Mat_DP oneda(ma,1), V(ma,ma);
		static Vec_DP atry(ma),beta(ma),da(ma), W(ma);

  if (alamda == -1){
    Mat_DP * aux = new  Mat_DP(ma,1);
    oneda = *aux;
    Mat_DP * aux1 = new  Mat_DP(ma,ma);
    V = *aux1;
    Vec_DP * aux2 = new  Vec_DP(ma);
    atry = *aux2;
    Vec_DP * aux3 = new  Vec_DP(ma);
    beta = *aux3;
    Vec_DP * aux4 = new  Vec_DP(ma);
    da = *aux4;
    Vec_DP * aux5 = new  Vec_DP(ma);
    W = *aux5;
  }

		if (alamda < 0.0) {
			mfit=0;
			for (j=0;j<ma;j++)
				if (ia[j]) mfit++;
			alamda=0.001;
			mrqcof(x,y,z,sig,a,ia,alpha,beta,chisq,funcs);
			ochisq=chisq;
			for (j=0;j<ma;j++) atry[j]=a[j];
		}
		Mat_DP temp(mfit,mfit);
		for (j=0;j<mfit;j++) {
			for (k=0;k<mfit;k++) covar[j][k]=alpha[j][k];
			covar[j][j]=alpha[j][j]*(1.0+alamda);
			for (k=0;k<mfit;k++) temp[j][k]=covar[j][k];
			oneda[j][0]=beta[j];
		}
  /*
 //delimitar os valores
   for(int i=0;i<ma-1;i+=9){
     if(a[i+0]<0)  a[i+0]=0;              //a
     if(a[i+0]>255)  a[i+0]=255;
     if(a[i+1]<0) a[i+1]=0;               //x
     if(a[i+1]>x.size()) a[i+1]=x.size();
     if(a[i+2]<0) a[i+2]=0;               //y
     if(a[i+2]>y.size()) a[i+2]=y.size();
     if(a[i+3]<0) a[i+3]=0;               //sx
     if(a[i+3]>50) a[i+3]=50;
     if(a[i+5]<0) a[i+5]=0;               //d
     if(a[i+5]>30) a[i+5]=30;
     //if(a[i+6]<0) a[i+6]=0;               //te
     //if(a[i+6]>50) a[i+6]=50;
     if(a[i+7]<0) a[i+7]=0;               //sy
     if(a[i+7]>50) a[i+7]=50;
     if(a[i+8]<0) a[i+8]=0;               //z0
     if(a[i+8]>255) a[i+8]=255;
   }
 //fim

		valGaussJ=gaussj(temp,oneda);
  if (!valGaussJ){
    Application->MessageBox("Error GaussJ","Error",MB_OK);
			alamda *= 10.0;
			chisq=ochisq;
   return;
  }*/

  svdcmp(temp,W,V);
		int wmax = 0.0;
		for (k=0;k<mfit;k++) wmax = max(wmax, W[k]);
		int wmin = wmax * 1.0e-6;
  for (k=0;k<mfit;k++) if (W[k] < wmin) W[k] = 0.0;
  svbksb(temp, W, V, beta, da);

		for (j=0;j<mfit;j++) {
			for (k=0;k<mfit;k++) covar[j][k]=temp[j][k];
			//da[j]=oneda[j][0];
		}
		if (alamda == 0.0) {
			covsrt(covar,ia,mfit);
			covsrt(alpha,ia,mfit);
			return;
```

```
		}
		for (j=0,l=0;l<ma;l++)
			if (ia[l]) atry[l]=a[l]+da[j++];
		mrqcof(x,y,z,sig,atry,ia,covar,da,chisq,funcs);
		if (chisq < ochisq) {
			alamda *= 0.1;
			ochisq=chisq;
			for (j=0;j<mfit;j++) {
				for (k=0;k<mfit;k++) alpha[j][k]=covar[j][k];
					beta[j]=da[j];
			}
			for (l=0;l<ma;l++) a[l]=atry[l];
		} else {
			alamda *= 10.0;
			chisq=ochisq;
		}
}
```

Mrqcof.Cpp

```
#include "nr.h"

void NR::mrqcof(Vec_I_DP &x, Vec_I_DP &y, Mat_I_DP &z, Vec_I_DP &sig, Vec_IO_DP &a,
		Vec_I_BOOL &ia, Mat_O_DP &alpha, Vec_O_DP &beta, DP &chisq,
		void funcs(const DP, const DP, Vec_I_DP &,DP &, Vec_O_DP &))
{
		int i,w,j,k,l,m,mfit=0;
		DP zmod,wt,sig2i,dz;

		int col=x.size();
  int lin=y.size();
		int ma=a.size();
		Vec_DP dzda(ma);
		for (j=0;j<ma;j++)		//+1
			if (ia[j]) mfit++;
		for (j=0;j<mfit;j++) {
			for (k=0;k<=j;k++) alpha[j][k]=0.0;
			beta[j]=0.0;
		}
		chisq=0.0;
  float maxdzda=0;
		for (i=0;i<lin;i++) {
		for (w=0;w<col;w++) {
			funcs(x[w],y[i],a,zmod,dzda);

			sig2i=1.0/(sig[w]*sig[w]);
			dz=z[w][i]-zmod;
    for (j=0,l=0;l<ma;l++) {
			if (ia[l]) {
       if(abs(dzda[l])>0.000001){
						wt=dzda[l]*sig2i;
         for (k=0,m=0;m<l+1;m++)
							if (ia[m]) alpha[j][k++] += wt*dzda[m];
         beta[j++] += dz*wt;
       }
       else j++;
      }
    }
			chisq += dz*dz*sig2i;
		}
  }
  for (j=1;j<mfit;j++)
			for (k=0;k<j;k++) alpha[k][j]=alpha[j][k];
}
```

fgauss.Cpp

```
#include <cmath>
#include "nr.h"
using namespace std;

inline int MyTrunc(float a){
  if (a>0)
    return floor(a);
  else
    return ceil(a);
}

void NR::fgauss(const DP x,const DP y, Vec_I_DP &a, DP &zn, Vec_O_DP &dzda)
{
        int i;
        DP fac,ex,arg;

        int na=a.size();
        zn=0.0;
  float A,B;
  float auxCos;
  float auxSen;
  float xf;
  float yf;
  float C;
  float a0,a1,a2,a3,a4,a5,a6,a7,a8;
  float aux1,aux2,aux3;

  for (i=0;i<na-1;i+=9){
    a0=a[i]; a1=a[i+1]; a2=a[i+2]; a3=a[i+3]; a4=a[i+4]; a5=a[i+5];
    a6=(a[i+6]-(MyTrunc(a[i+6]/(2*M_PI))))*(2*M_PI);
    a7=a[i+7]; a8=a[i+8];

    if((x!=a1)&&(y!=a2)){
      auxCos = cos(a6);
      auxSen = sin(a6);
      xf=x-a1;
      yf=y-a2;
      C=2/a5;
      A=(xf*auxCos+yf*auxSen)/a3;
      B=(-xf*auxSen+yf*auxCos)/a7;

      zn+=a0*exp(-0.5*std::pow((A*A)+(B*B),C));

      aux1=a0*exp(-0.5*std::pow((A*A)+(B*B),C));
      aux3=std::pow(std::pow(B,2)+std::pow(A,2),(C-1));
/*a*/           dzda[i]=aux1/a0;
/*x0-*/     dzda[i+1]=(-1/a5)*(aux1*(((2*auxSen*B)/(a7))-((2*auxCos*A)/(a3)))*aux3);
/*y0-*/         dzda[i+2]=(-1/a5)*(aux1*((-(2*auxCos*B)/(a7))-((2*auxSen*A)/(a3)))*aux3);
/*Sx*/      dzda[i+3]=(1/(a5*std::pow(a3,3)))*(2*aux1*std::pow(A*a3,2)*aux3);
/*Sf*/      dzda[i+4]=0;
/*d   log(-0.5..*/                  //   dzda[i+5]=(-2/(a5*a5))*aux1*log(0.5*(std::pow(A,2)+std::pow(B,2)))*(-
0.5)*std::pow((B*B)+(A*A),C);
/*te-*/     dzda[i+6]=(-1/a5)*(aux1*(((2*(B*a7)*(-A*a3))/(a7*a7))+((2*B*a7*A*a3)/(a3*a3)))*aux3);
/*Sy*/      dzda[i+7]=(1/(a5*std::pow(a7,3)))*(2*aux1*std::pow(B*a7,2)*aux3);
/*z0*/      dzda[i+8]=1;
    }
  }
  zn+=a8;
}
```

Anexo III - Algoritmo Levenberg-Marquardt