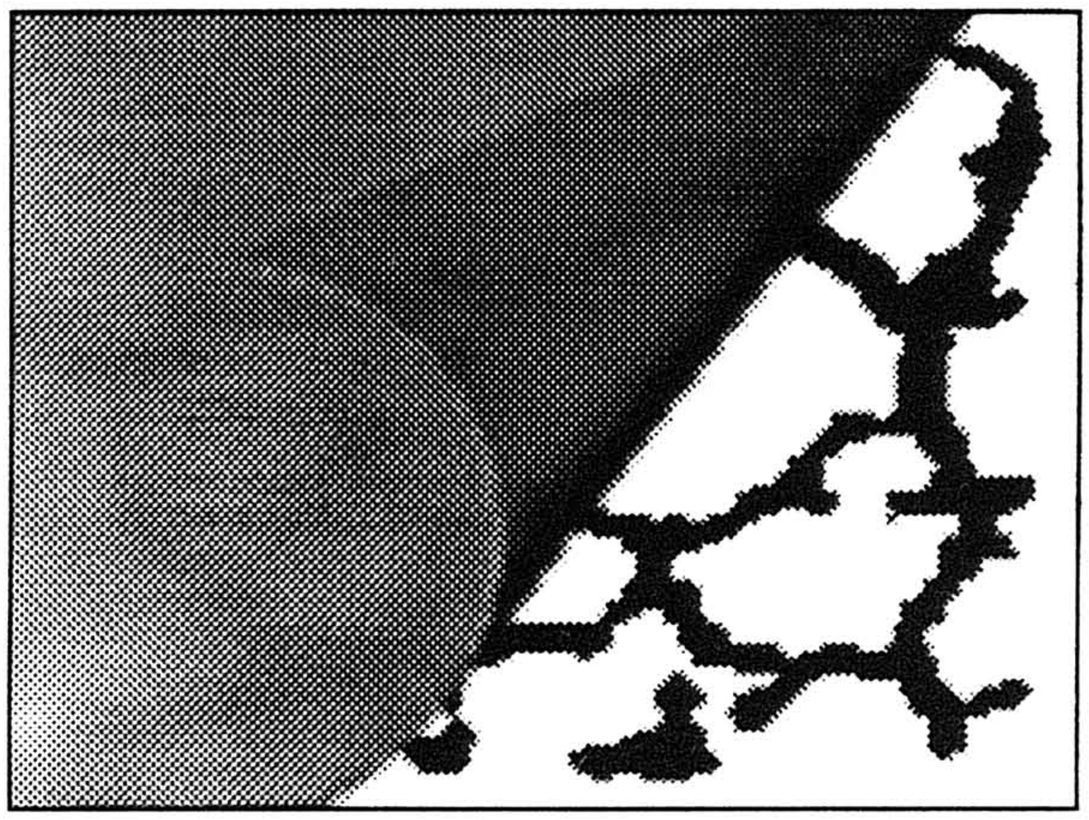
Faculdade de Engenharia da Universidade do Porto



Processamento e Análise de Imagens Histeroscópicas Uterinas

Detecção de redes vasculares

Hélio Mendonça

DEEC
FEUP

Dissertação no âmbito do
Curso de Mestrado em Engenharia Electrotécnica e de Computadores
Área de especialização em Informática Industrial
Edição de 1991/92

FEUP

AGRADECIMENTOS

Gostaria de agradecer ao Prof. Marques de Sá pelo apoio e incentivo dados ao longo deste trabalho, ao Dr. Silva Carvalho pela colaboração de carácter médico que prestou e ainda à JNICT pela atribuição da bolsa de estudo BM/1850/91 - IA.

Este trabalho é dedicado aos meus pais por me terem posto no mundo e à minha namorada que quando necessário me tira dele.



cat
681.3(043) | MEN fh | PRO
U.P. Faculdade de Engenharia
BIBLIOTECA 1
N.º 21520
CDU
Data 17 4 1996
N.º: 2721

Porto, 28 de Janeiro de 1994

ÍNDICE

Prefácio iv

1 Introdução 1
1.1 Objectivo 2
1.2 Imagens 4

2 Detecção de redes vasculares 7
2.1 Introdução 8
2.2 Thresholding 9
2.3 Edge detection 10
2.4 Vessel tracking 11

3 Método desenvolvido 12
3.1 Descrição 13
3.2 Pré-processamento 14
3.2.1 Remoção da superfície base 15
3.2.2 Equalização 18
3.3 Processamento 19
3.3.1 Detecção de mínimos locais 20
3.3.2 Thinning 22
3.3.3 Decomposição em segmentos 23
3.3.4 Classificação dos segmentos 24
3.3.5 Reconstrução da rede vascular 25
3.4 Análise 26
3.4.1 Extracção de características 27

4 Sistema 29
4.1 Hardware 36
4.2 Software 31

5 Resultados 33
5.1 Imagens simuladas 34
5.2 Imagens reais 36

6 Conclusões 38

7 Bibliografia 41

Apêndice A: Manual do programa

Apêndice B: Listagens do programa

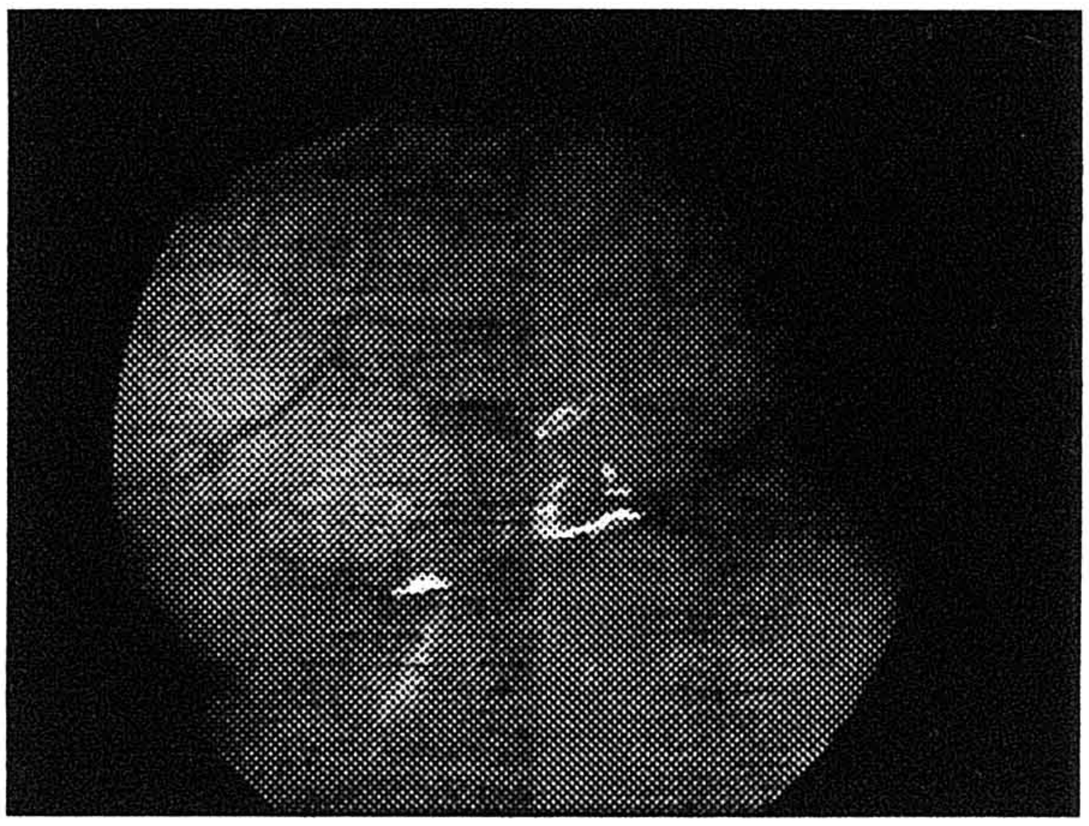
Apêndice C: Artigo

PREFÁCIO

Esta dissertação insere-se no Curso de Mestrado em Engenharia Electrotécnica e de Computadores, área de especialização em Informática Industrial, que decorreu na Faculdade de Engenharia da Universidade do Porto na edição de 1991/92.

O trabalho foi proposto pelo Serviço de Ginecologia do Hospital de S. João do Porto através do Dr. J. L. Silva Carvalho e teve como principal orientador o Prof. J. P. Marques de Sá.

A estrutura da dissertação será a seguinte. No capítulo 1 é apresentado o objectivo do trabalho assim como as imagens histeroscópicas uterinas. No capítulo 2 será discutida a operação de detecção de redes vasculares (fase crucial de todo o processamento) sendo também analisados alguns métodos existentes neste domínio. O capítulo 3 destina-se a descrever o algoritmo de processamento e análise desenvolvido e que inclui um método alternativo de detecção de redes vasculares. O sistema que implementa este algoritmo será apresentado no capítulo 4 quer na sua vertente de hardware quer na de software sendo os seus resultados analisados no capítulo 5. O capítulo 6 é destinado à indicação de conclusões e aspectos passíveis de sofrerem futuras evoluções. A bibliografia utilizada é listada no capítulo 7. Em apêndice é ainda possível consultar o manual do programa desenvolvido (SPIMUT), as listagens do código fonte e ainda o artigo submetido à GKPO'94 sobre este trabalho.



CAPÍTULO 1

INTRODUÇÃO

Neste capítulo será indicado o objectivo deste trabalho sendo também apresentadas as imagens histeroscópicas uterinas que constituem o alvo de todo o processamento.

1.1 Objectivo

As redes vasculares desempenham um papel primordial no correcto funcionamento dos tecidos e orgãos por elas irrigados. A rede vascular uterina situa-se no interior da mucosa que cobre toda a parede interna do útero. Esta mucosa designada por endométrio (do grego ενδον "dentro", e μητρα "útero") passa por diferentes fases de desenvolvimento ao longo do ciclo menstrual.

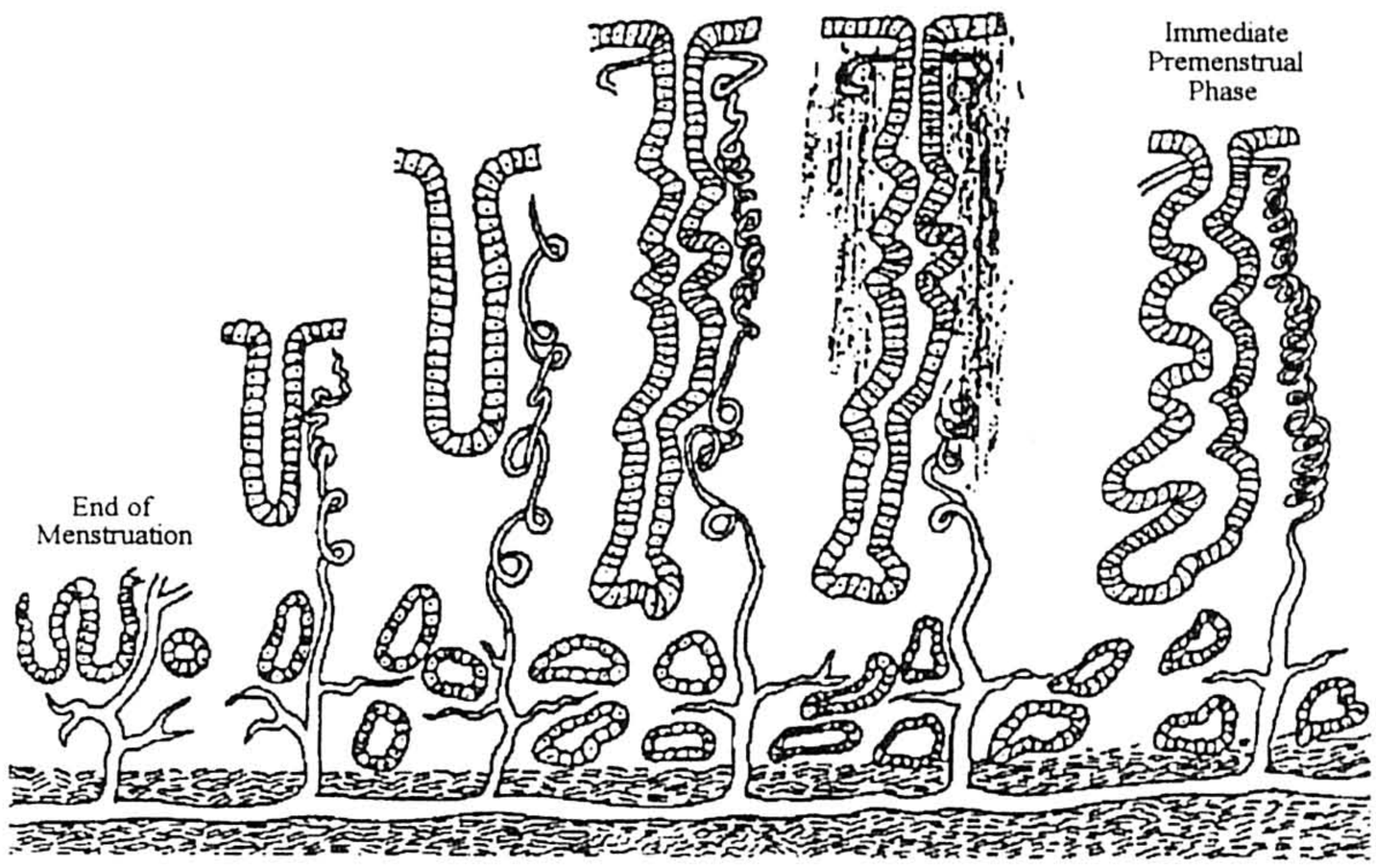


Fig. 1 - Endométrio ao longo do ciclo menstrual

Tal como a Fig. 1 mostra a rede vascular é praticamente inexistente na fase pós--menstrual sendo extremamente densa na fase que antecede a menstruação. Este desenvolvimento pode contudo ser deficiente ou excessivo denotando situações de infertilidade e cancro respectivamente. Prova deste facto são as Figs. 2a e 2b onde se podem ver panorâmicas do endométrio de duas pacientes numa mesma fase do ciclo. Enquanto que a primeira apresenta uma rede com um normal desenvolvimento na segunda a rede praticamente não existe indiciando sintomas de infertilidade.

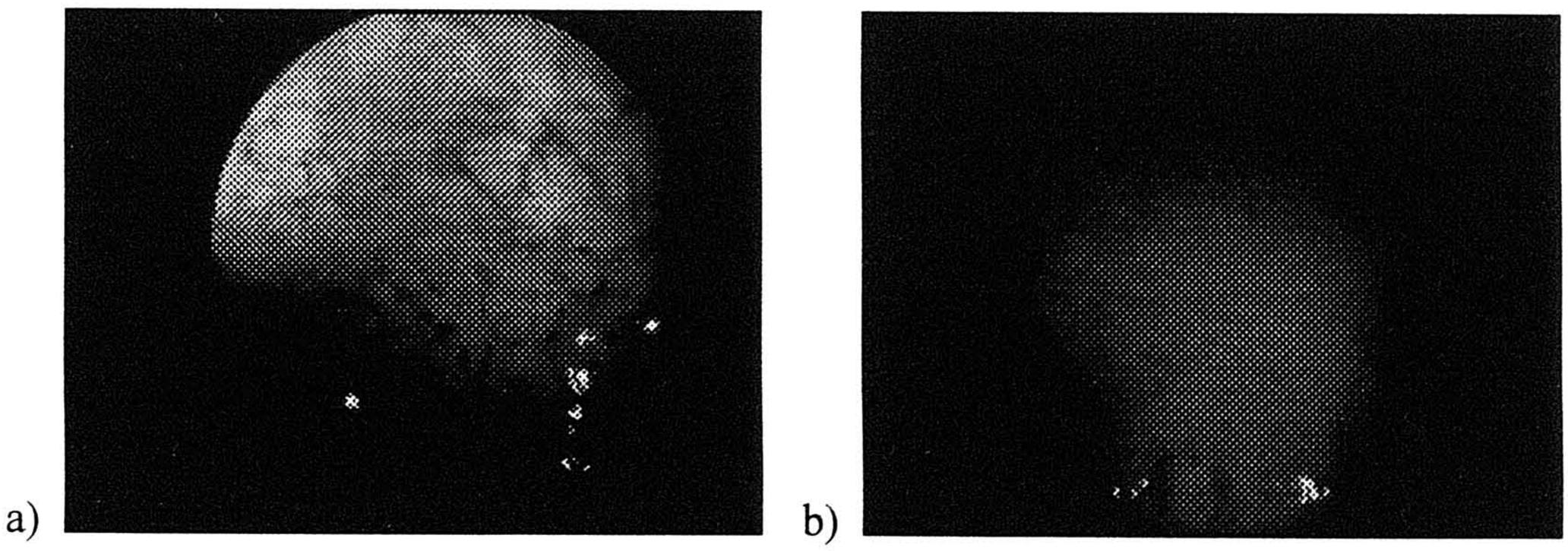


Fig. 2 - Desenvolvimento do endométrio: a) normal, b) deficiente

O Dr. J. L. Silva Carvalho no âmbito da sua tese de doutoramento efectua no Serviço de Ginecologia do Hospital de S. João do Porto, um estudo minucioso sobre o papel desempenhado pela rede vascular uterina em determinadas situações patológicas como a infertilidade e o cancro. Este estudo, tendo por base a análise histeroscópica do útero, só é contudo fiável se acompanhado por uma caracterização objectiva, quantitativa e precisa da rede vascular, algo que a simples inspecção visual das imagens não permite.

O objectivo deste trabalho é pois a criação de um sistema computacional de processamento e análise de imagens histeroscópicas uterinas que proceda à desejada caracterização da rede vascular. Os valores tomados por essas características serão guardados em disco para uma posterior análise estatística.

Embora existam vários exemplos da aplicação do processamento de imagem na caracterização e análise de redes vasculares (imagens angiográficas, imagens do fundo do olho, etc) é a primeira vez que ele se aplica a imagens histeroscópicas com todas as especificidades e dificuldades por elas apresentadas.

1.2 Imagens

As imagens a processar são obtidas por histeroscopia. Histeroscopia é a inspecção da cavidade uterina usando para tal um endoscópio de fibra óptica (histeroscópio). O histeroscópio dá uma vista panorâmica do canal cervical e da cavidade uterina, sendo suficiente para a maioria dos fins de diagnóstico.

O histeroscópio utilizado (Hamou tipo II), tal como a Fig. 3 mostra, é constituído por uma sonda de 4 mm de diâmetro onde se encontram alinhadas um conjunto de lentes de Rod. Essa sonda é envolvida por uma baínha pela qual passa o gás que procede à distensão do útero (operação imprescindível à obtenção de uma visão clara). Este gás (dióxido de carbono) é injectado a uma taxa de 100 ml/min a pressões não superiores a 100 mmHg. Na extremidade A dessa sonda é acoplada a câmara de captação de imagem. A iluminação é efectuada pelo próprio histeroscópio sendo para tal utilizada uma fonte de luz fria regulável de 300W que é encaminhada até ao ponto B por fibra óptica. Este aparelho possibilita ainda factores de ampliação até 80 x por ajuste do parafuso C.

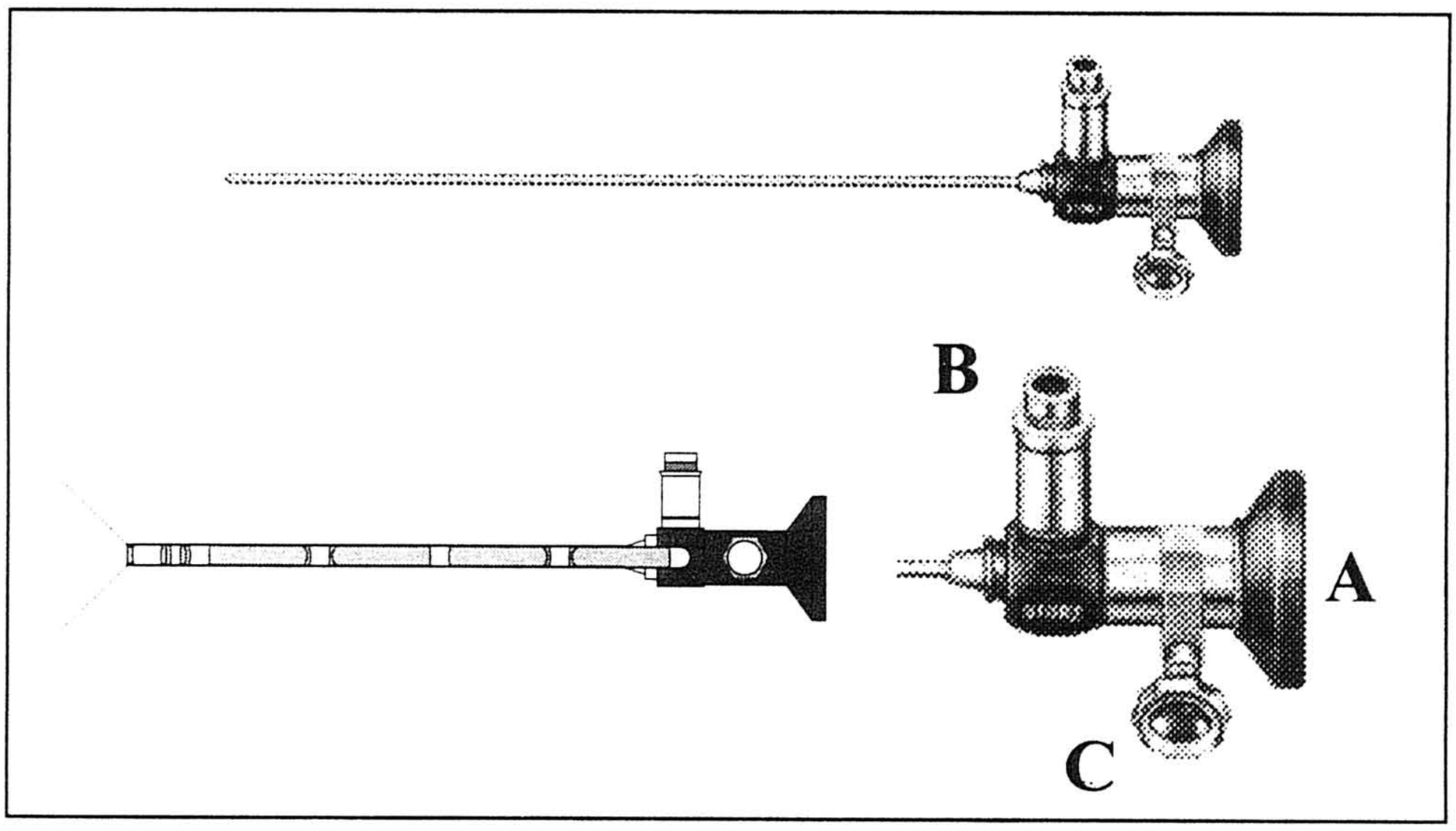


Fig. 3 - Histeroscópio

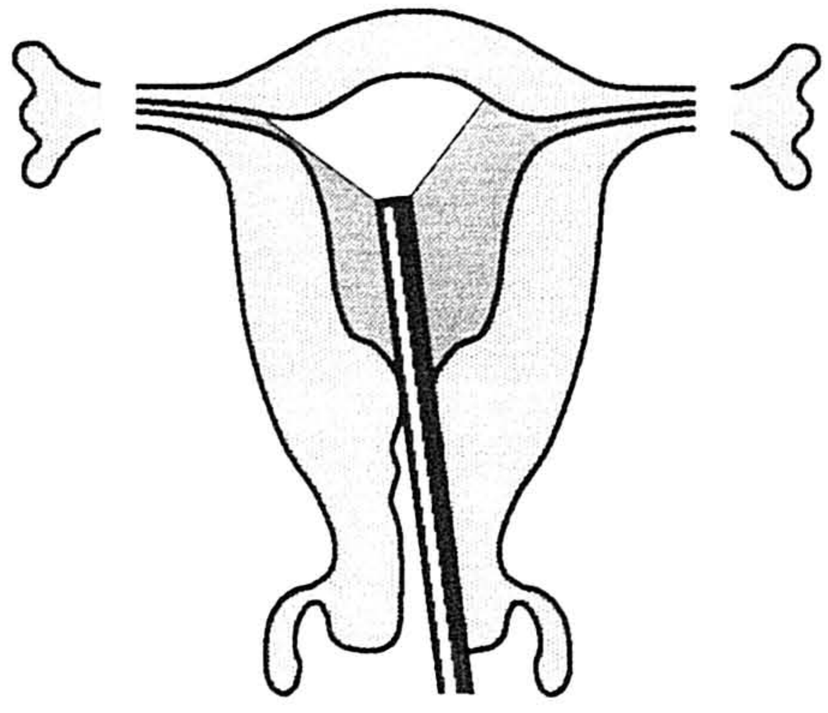


Fig. 4 - Captação das imagens

O histeroscópio é introduzido pela abertura externa sendo avançado lentamente ao longo do canal cervical. A cavidade uterina surge depois de ultrapassada a abertura interna. O fundo do útero fica então visível mostrando o endométrio com a respectiva rede vascular que irá constituir o alvo de todo o processamento. A captação das imagens faz-se a alguns milímetros da parede uterina (Fig. 4).

As imagens histeroscópicas captadas durante consultas de rotina efectuadas no Serviço de Ginecologia do Hospital de S. João do Porto são gravadas em cassetes de vídeo. Posteriormente estas imagens são digitalizadas por meio de uma placa de aquisição de imagem que integra o sistema de processamento desenvolvido. As imagens digitalizadas têm uma resolução de 512 por 512 pixels e são monocromáticas (256 tons de cinzento). Contudo apenas uma região de interesse (ROI) a seleccionar oportunamente será efectivamente processada em virtude do seu maior significado clínico.

De seguida é apresentada uma imagem histeróscopica uterina assim como a região de interesse seleccionada. De notar que esta ROI irá ser constantemente utilizada nos próximos capítulos para exemplificar todas as operações de processamento.

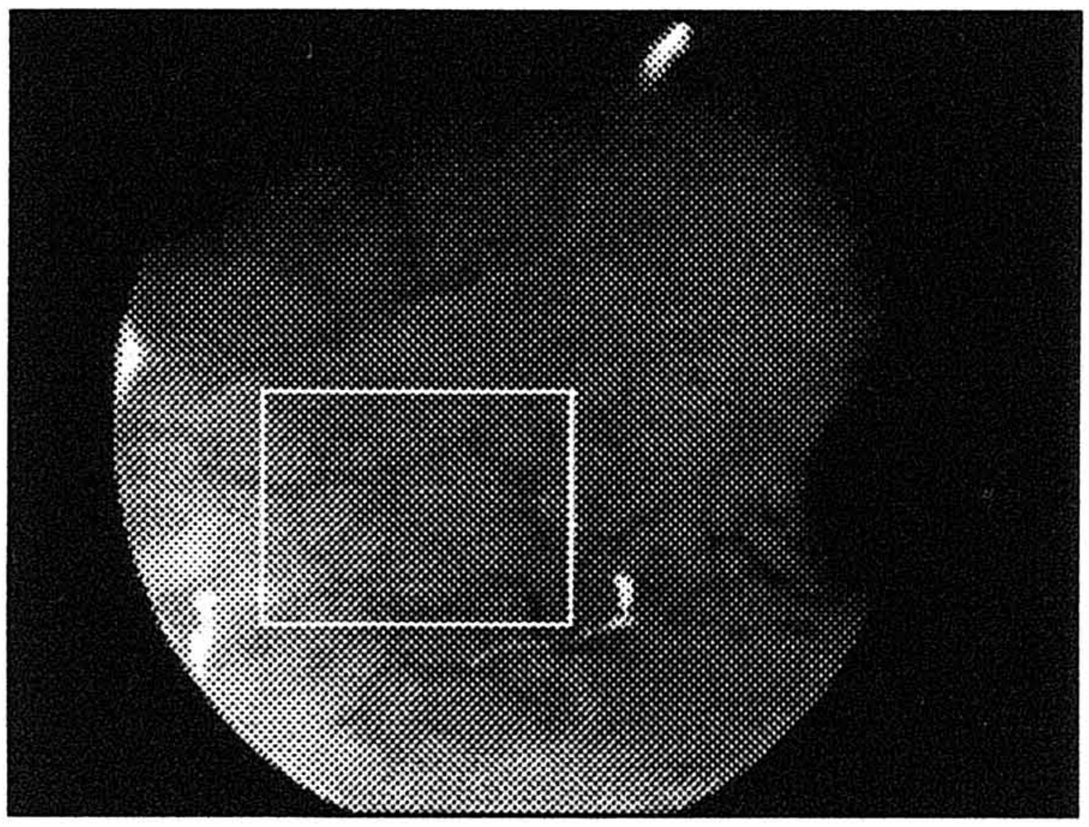


Fig. 5 - Imagem histeroscópica uterina



Fig. 6 - Região de interesse (ROI)

Como facilmente se imagina pela descrição já efectuada o processo de captação de imagens é bastante desconfortável para a paciente. Este facto leva a que a referida captação seja bastante fugaz e consequentemente pouco cuidada em termos de focagem e iluminação.

Adicionalmente verifica-se a presença de artefactos nas imagens tais como coágulos de sangue, bolhas criadas pelo gás injectado e ainda reflexos provocados na parede uterina pela fonte luminosa.

Como já foi referido, em certas fases do ciclo menstrual a mucosa uterina (endométrio) encontra-se bastante desenvolvida. O facto dos vasos situarem-se no interior dessa mucosa que apresenta um aspecto translúcido torna-os pouco contrastados e bastante esbatidos em termos visuais. O nível de ruído é também significativo.

Todos estes factores, alguns deles perfeitamente observáveis nas imagens seguintes, dificultam enormemente o processamento em geral e a operação de detecção da rede vascular em particular.

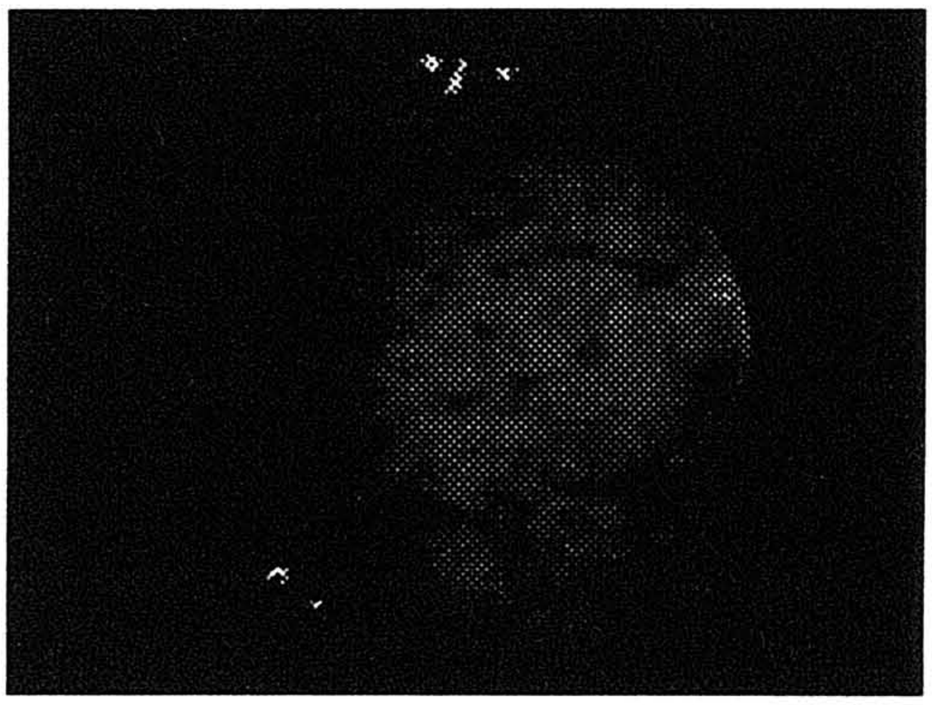


Fig. 7 - Deficiente iluminação

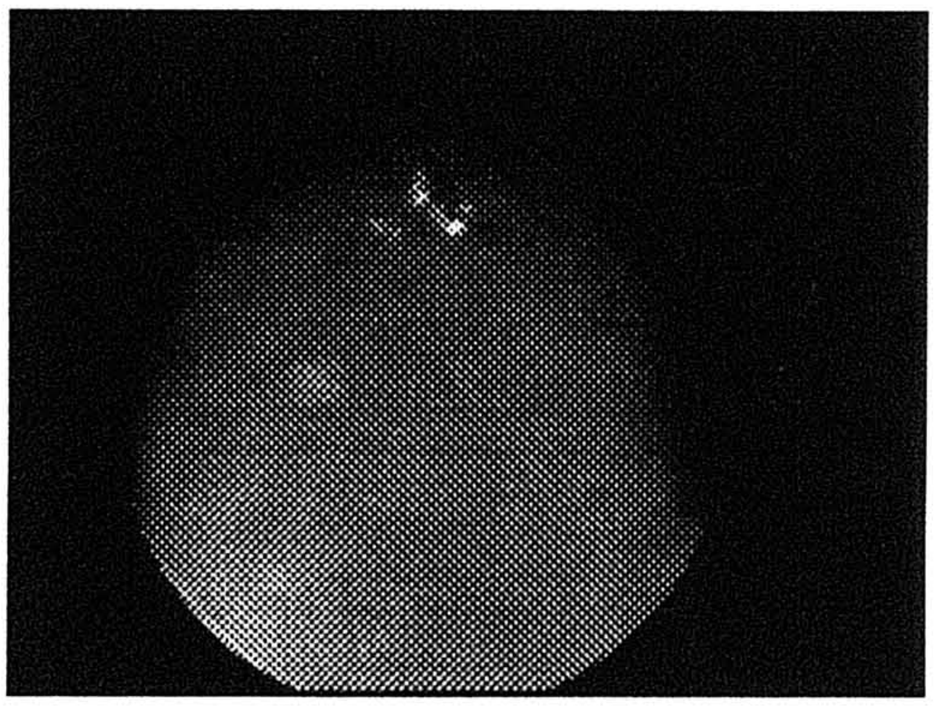


Fig. 8 -Deficiente focagem

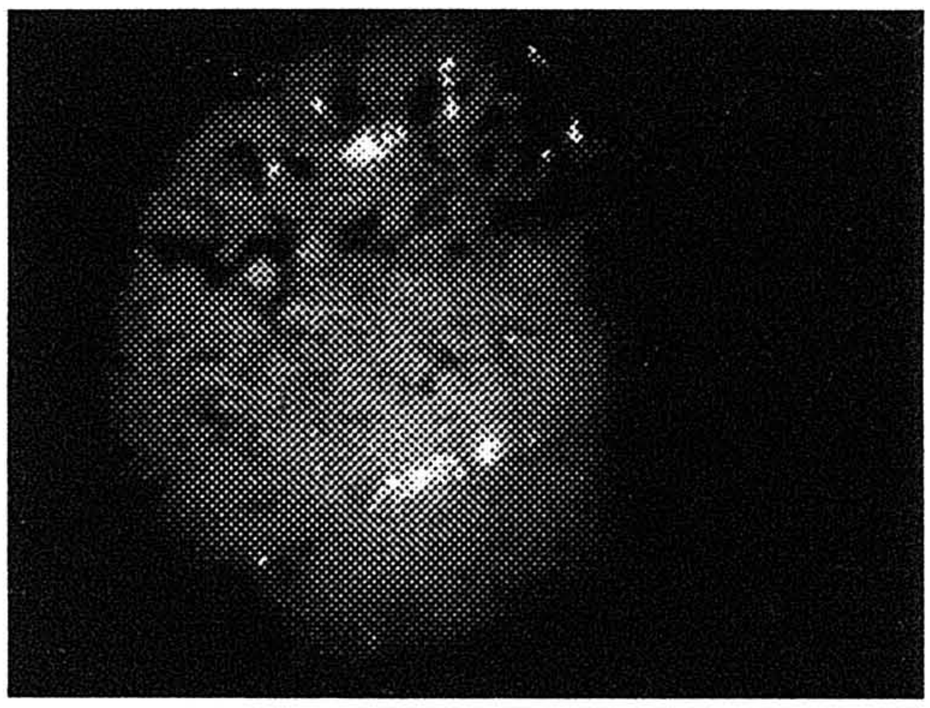


Fig. 9 - Coágulos de sangue

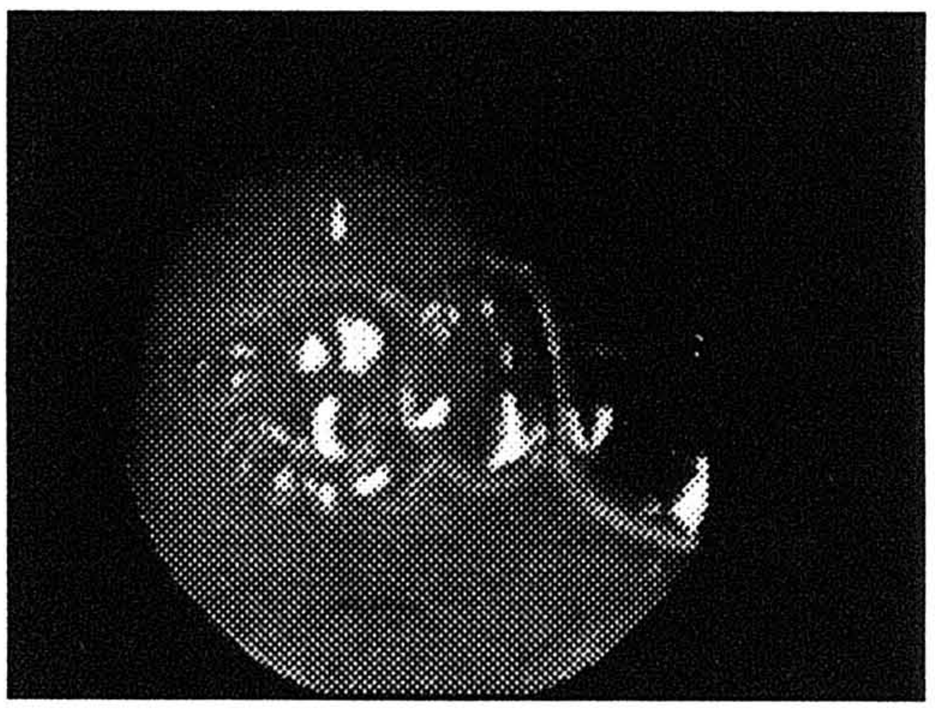


Fig. 10 - Bolhas criadas pelo gás

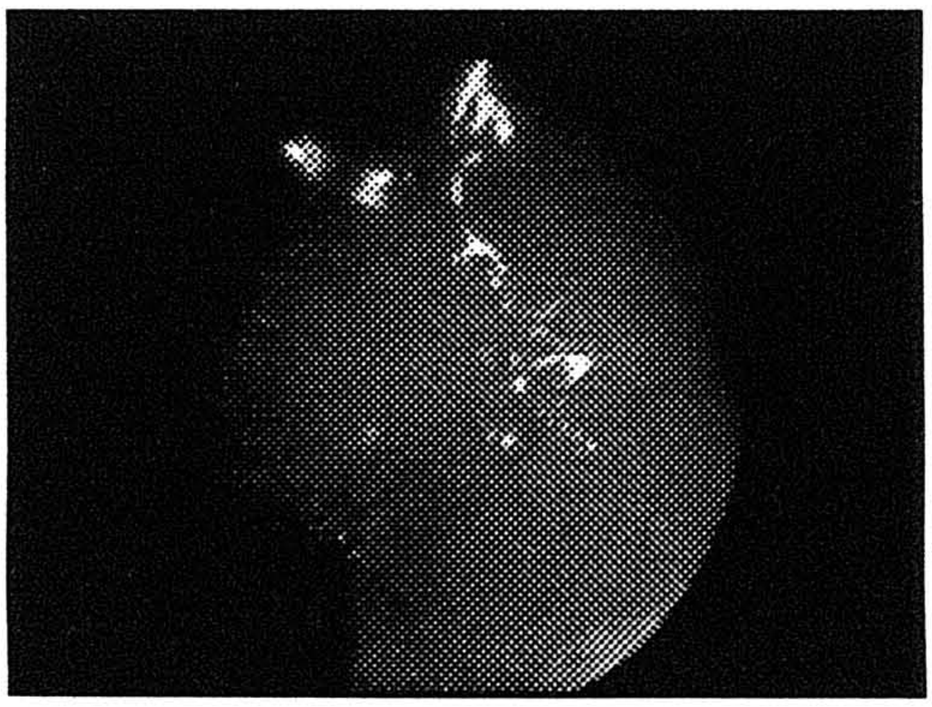


Fig. 11 - Reflexos

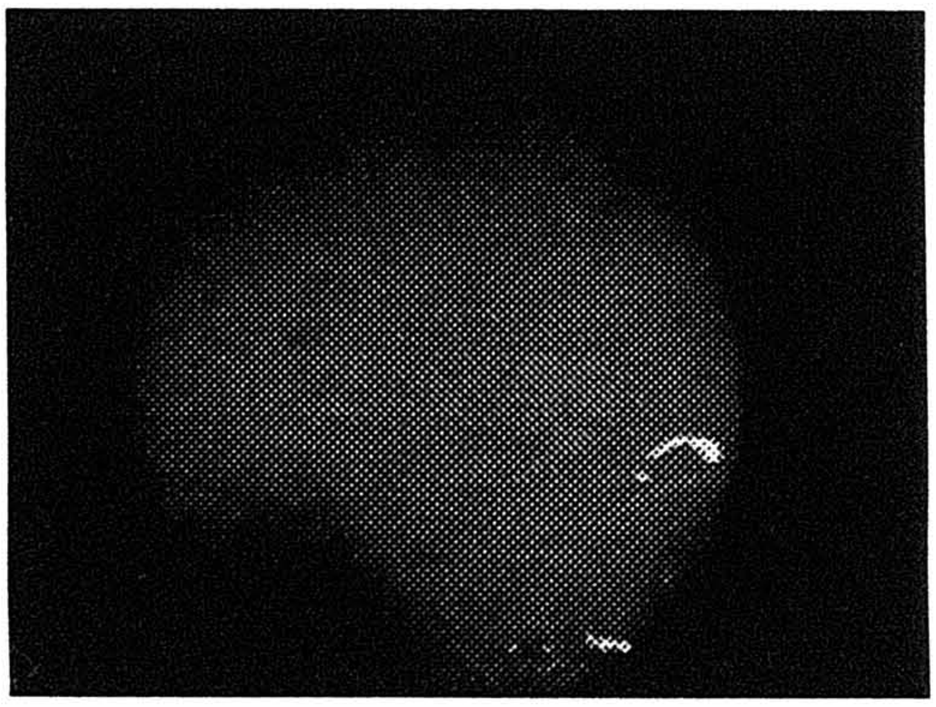
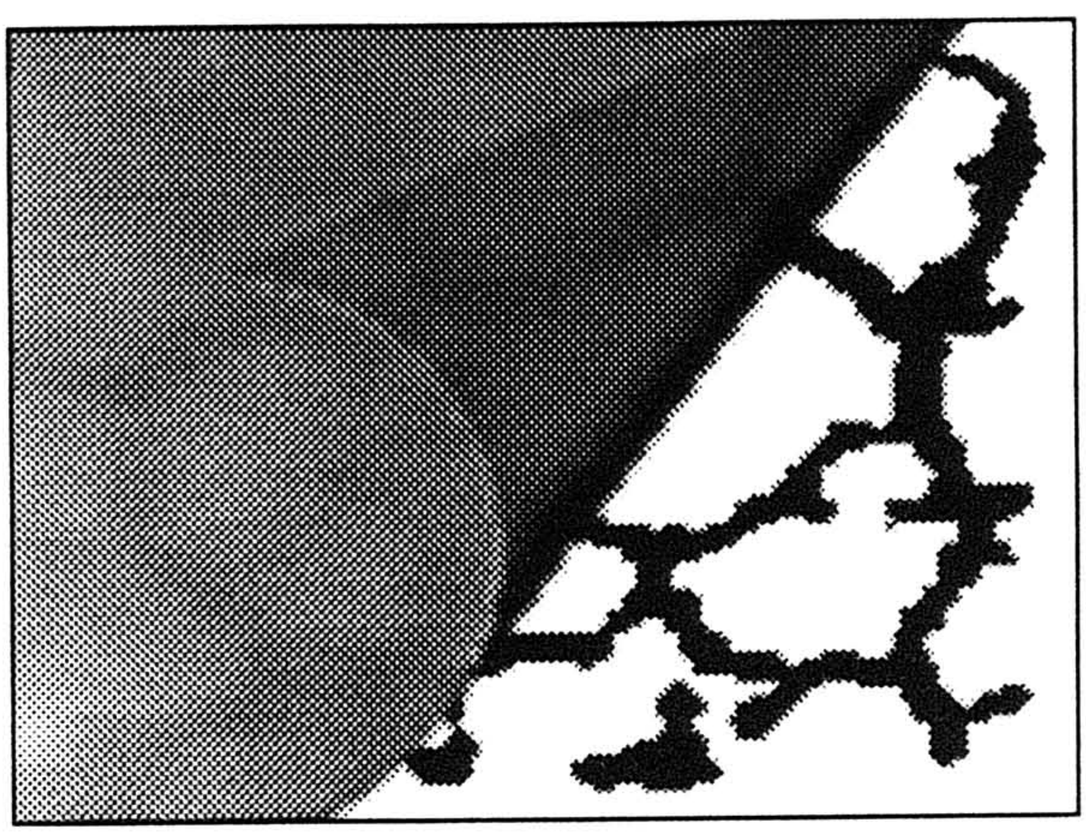


Fig. 12 - Vasos pouco contrastados



CAPÍTULO 2

DETECÇÃO DE REDES VASCULARES

Neste capítulo será analisado o processo de detecção de redes vasculares (fase crucial de todo o processamento). Serão também apresentados alguns métodos existentes que executam a referida detecção.

2.1 Introdução

O processo de detecção de redes vasculares é basicamente um processo de segmentação. Com ele pretende-se decidir de entre os pixels constituintes da imagem quais são os que pertencem à rede vascular e quais são os que pelo contrário pertencem ao fundo da imagem. O objectivo é pois o de obter uma imagem binária (preto e branco) definidora da rede a partir de uma imagem monocromática.

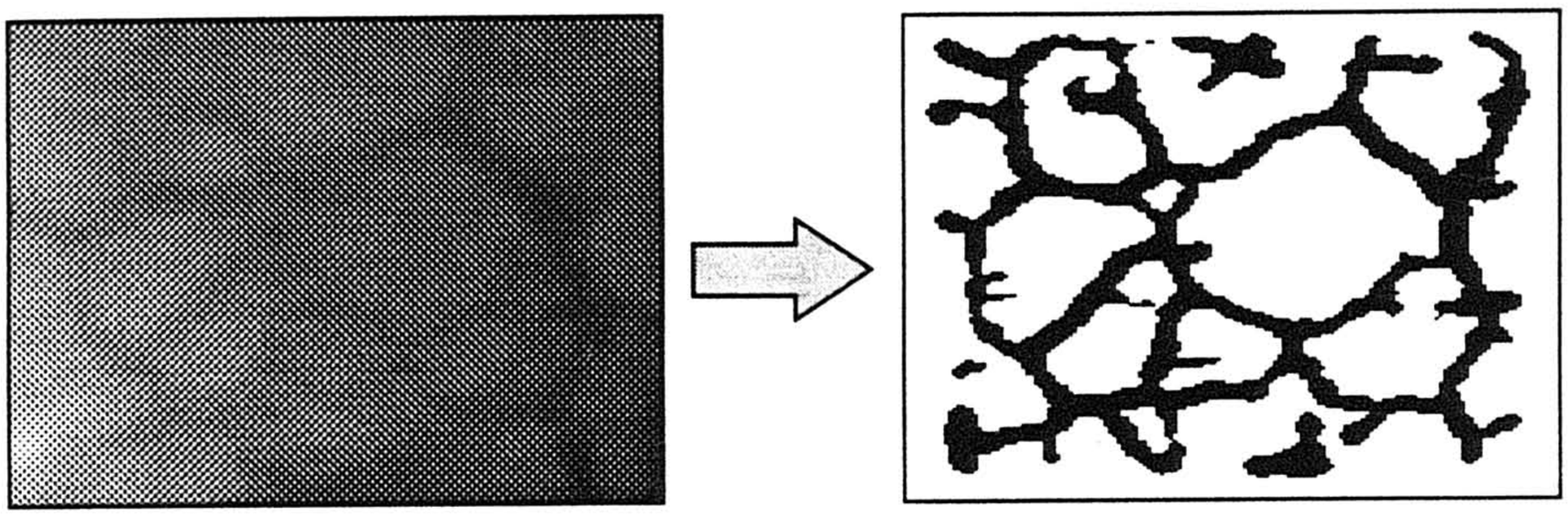


Fig. 13 - Detecção da rede vascular

Os vasos constituintes das redes vasculares possuem determinadas características que os identificam no seio de uma imagem. Essas características são basicamente as seguintes:

C1 A intensidade dos vasos é de uma forma geral menor do que a do fundo da imagem. Este facto deve-se à menor reflectância destes quando comparada com a do fundo da imagem.

C2 Os vasos possuem uma forma alongada sendo delimitados por arestas aproximadamente paralelas e onde se verificam transições de intensidade mais ou menos significativas.

C3 Os vasos organizam-se numa rede sob a forma de malha ou árvore apresentando uma forte continuidade.

Nas próximas secções deste capítulo serão descritos alguns dos métodos mais utilizados para efectuar a detecção de redes vasculares. Estes métodos exploram algumas das características acima referidas.

2.2 Thresholding

O método designado por "thresholding" é bastante simples e rápido. Neste método é explorada a primeira das características anteriormente enunciadas (*C1*). Os pixels da imagem são classificados como pertencentes à rede vascular se a sua intensidade se situar abaixo de um limiar de intensidade (threshold). Este limiar pode ser determinado automaticamente, por inspecção do histograma da imagem.

Os resultados da aplicação deste método podem ser inaceitáveis caso a intensidade do fundo da imagem não seja constante por questões de deficiente iluminação por exemplo. Nestas condições é impossível definir um limiar válido para toda a imagem, como se pode constatar na Fig. 14.

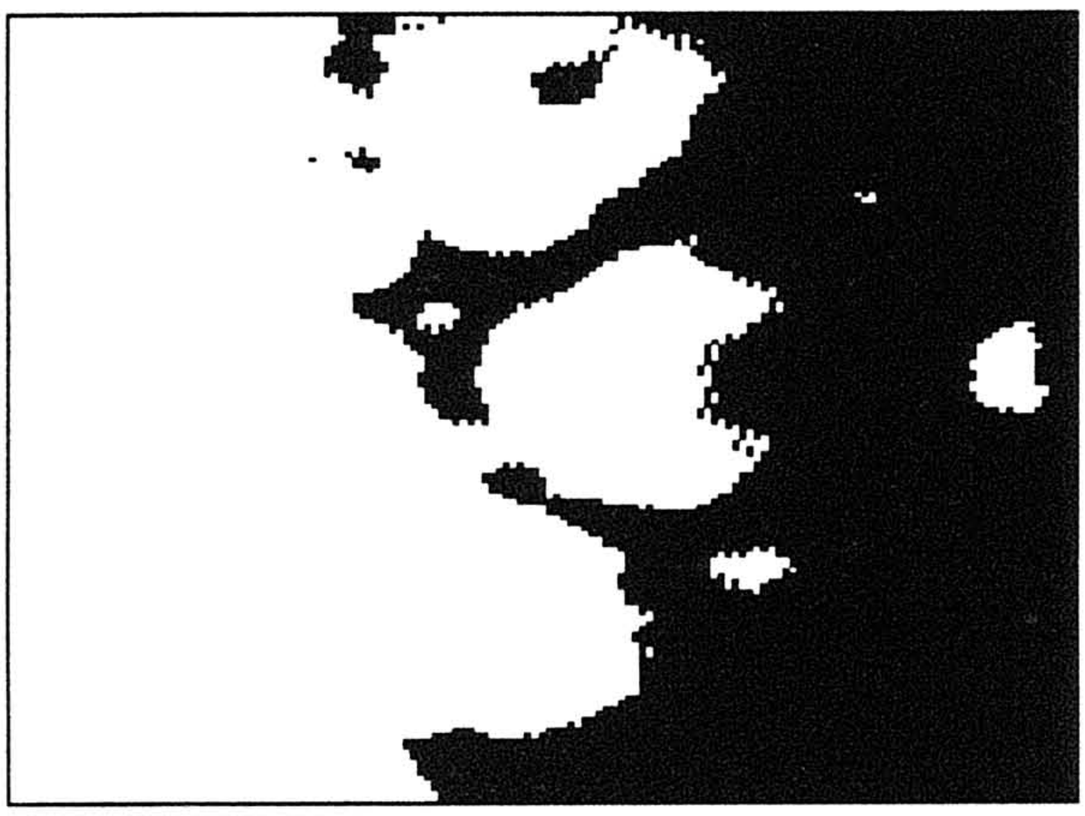


Fig. 14 - Thresholding

Mesmo que o fundo da imagem possua uma intensidade constante o problema subsiste caso a rede seja composta por vasos de diferente intensidade. A escolha de um limiar adequado para a detecção de um vaso pode levar ao desaparecimento de um outro como se ilustra nas Figs. 15a e 15b.

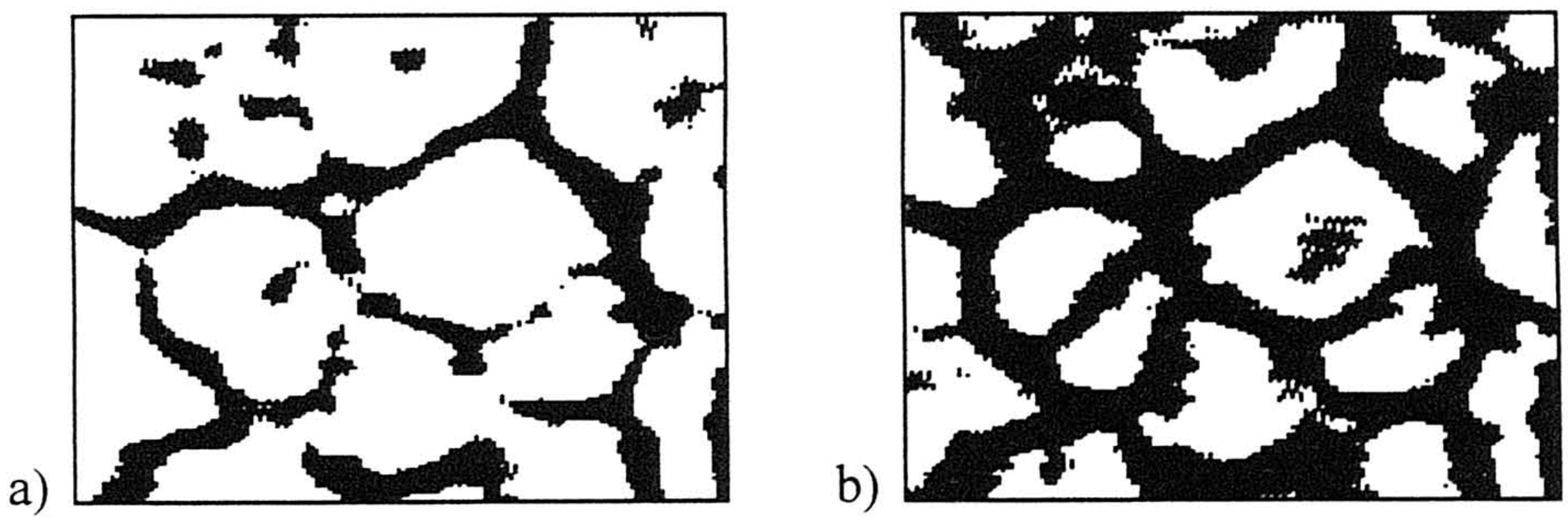


Fig. 15 - Thresholding: a) Limiar A, b) Limiar B

No caso particular das imagens histeroscópicas ocorrem ambas estas situações. A intensidade do fundo da imagem não é constante devido quer à geometria do orgão quer ao tipo de iluminação utilizado. Por seu lado os vasos podem possuir diferentes intensidades consoante a profundidade a que estes se encontram na mucosa uterina.

2.3 Edge detection

O método de detecção de redes vasculares designado por "edge detection" explora basicamente a segunda das características dessas redes (*C2*). Neste método procura-se localizar as arestas que delimitam os vasos. Para tal utilizam-se filtros detectores de arestas, de que o Laplaciano é o exemplo mais simples, ou processos mais elaborados como os de Canny e Deriche.

O resultado da aplicação deste método nem sempre é satisfatório, principalmente quando as imagens são bastante ruidosas e possuem arestas mal definidas. As imagens histeroscópicas como já foi referido são bastante esbatidas pelo que as transições de intensidade entre os vasos e o fundo da imagem são muito suaves e consequentemente inadequadas para serem localizadas de forma precisa por este método.

Como se pode constatar nas Figs. 17a e 17b mesmo aplicando um método de detecção de arestas bastante refinado como o de Canny ou Deriche obtém-se uma imagem final extremamente confusa.

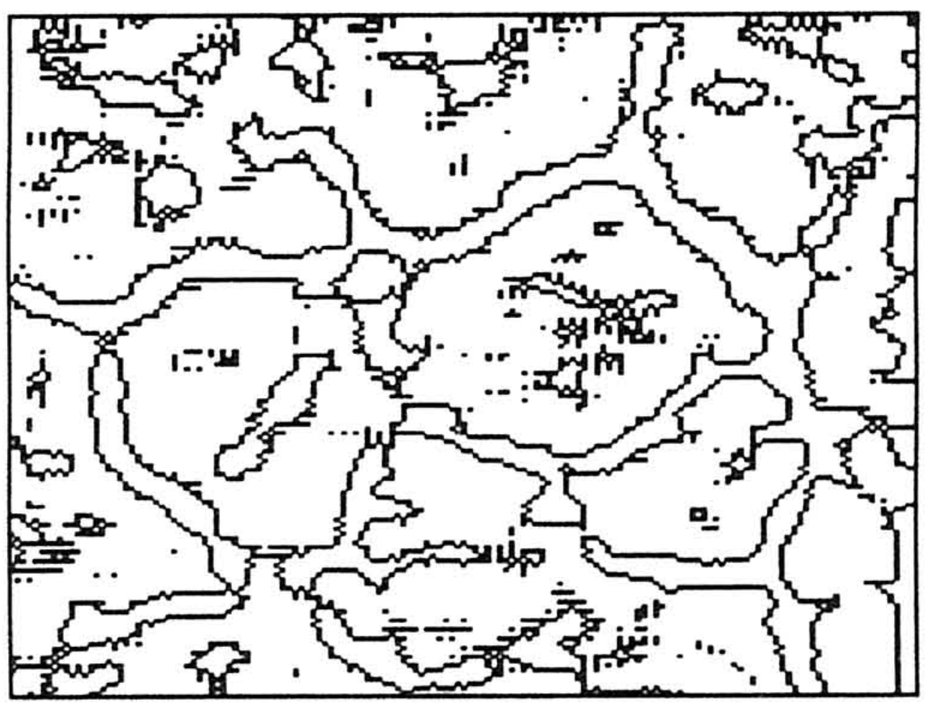
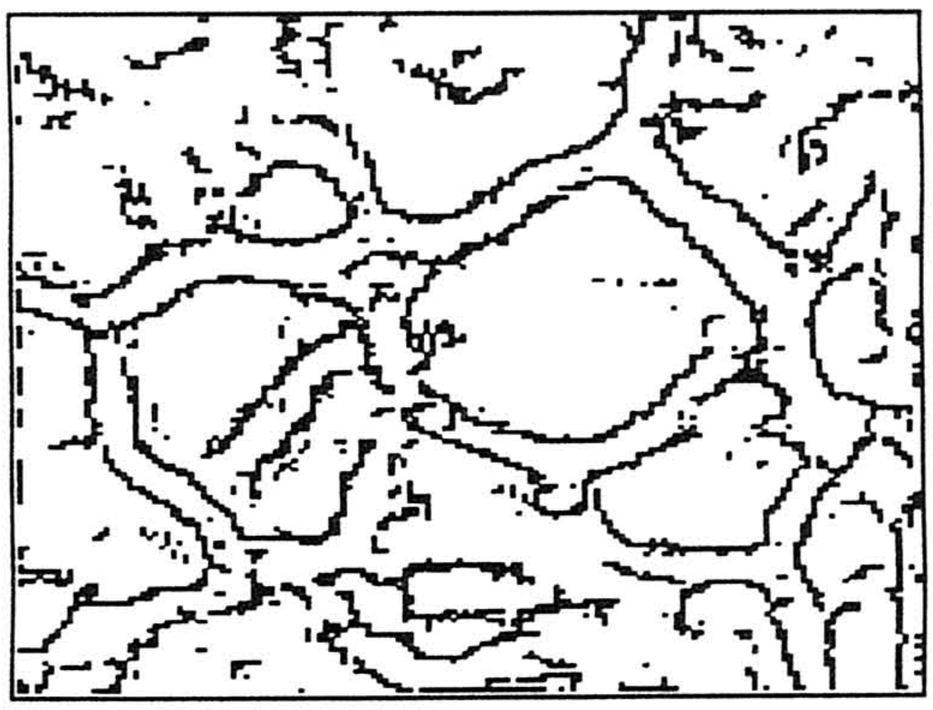


Fig. 16 - Edge detection: Laplaciano

a)



b)

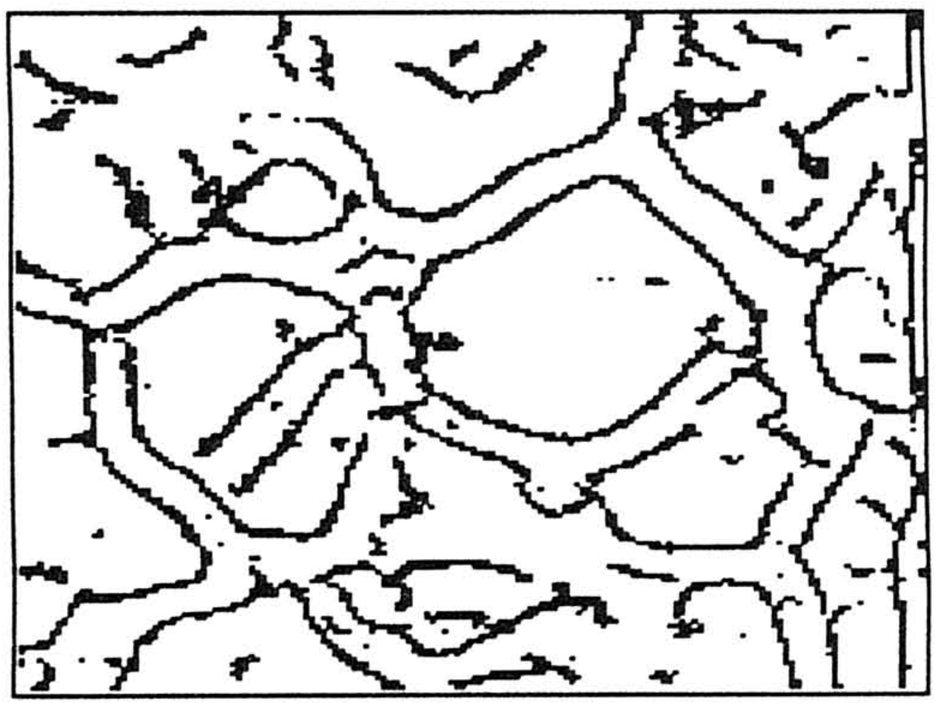


Fig. 17 - Edge detection: a) Método de Canny, b) Método de Deriche

2.4 Vessel tracking

O terceiro método aqui apresentado designa-se "vessel tracking" e como o nome indica percorre os vasos explorando assim a característica de continuidade (*C3*) apresentada pelas redes vasculares. Trata-se pois de um método iterativo em que partindo de um ponto inicial pertencente à rede e por análise da direcção tomada pelo vaso se vai encontrando o ponto seguinte.

Este método apresenta melhores performances quando a rede se organiza na forma de árvore uma vez que não existem percursos fechados como ocorreria no caso de uma estrutura em malha.

Uma vez que este método baseia o seu funcionamento na característica de continuidade apresentada pela rede é essencial que esta se verifique. No caso das imagens histeroscópicas como já foi referido esta situação nem sempre acontece uma vez que os vasos podem desaparecer sob a mucosa. A aplicação deste método a estas imagens, tal como se vê na Fig. 18, pode levar a resultados errados uma vez que, por várias vezes, o método perde a pista do vaso.

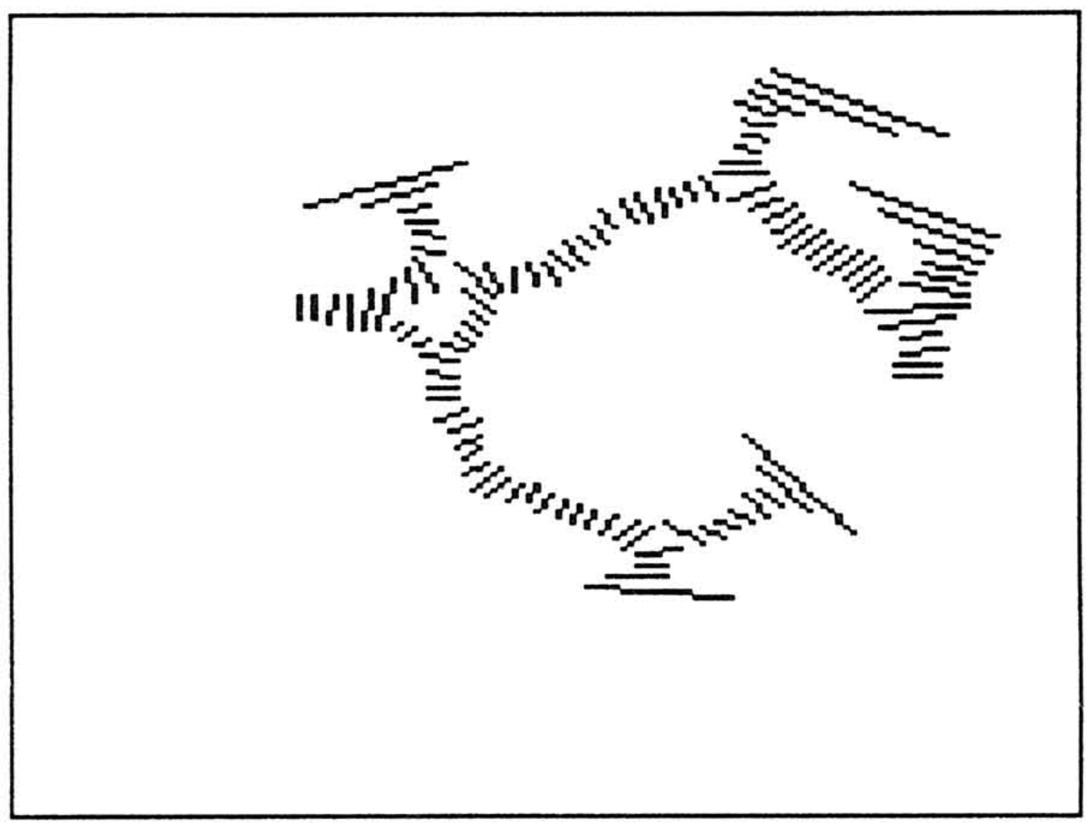
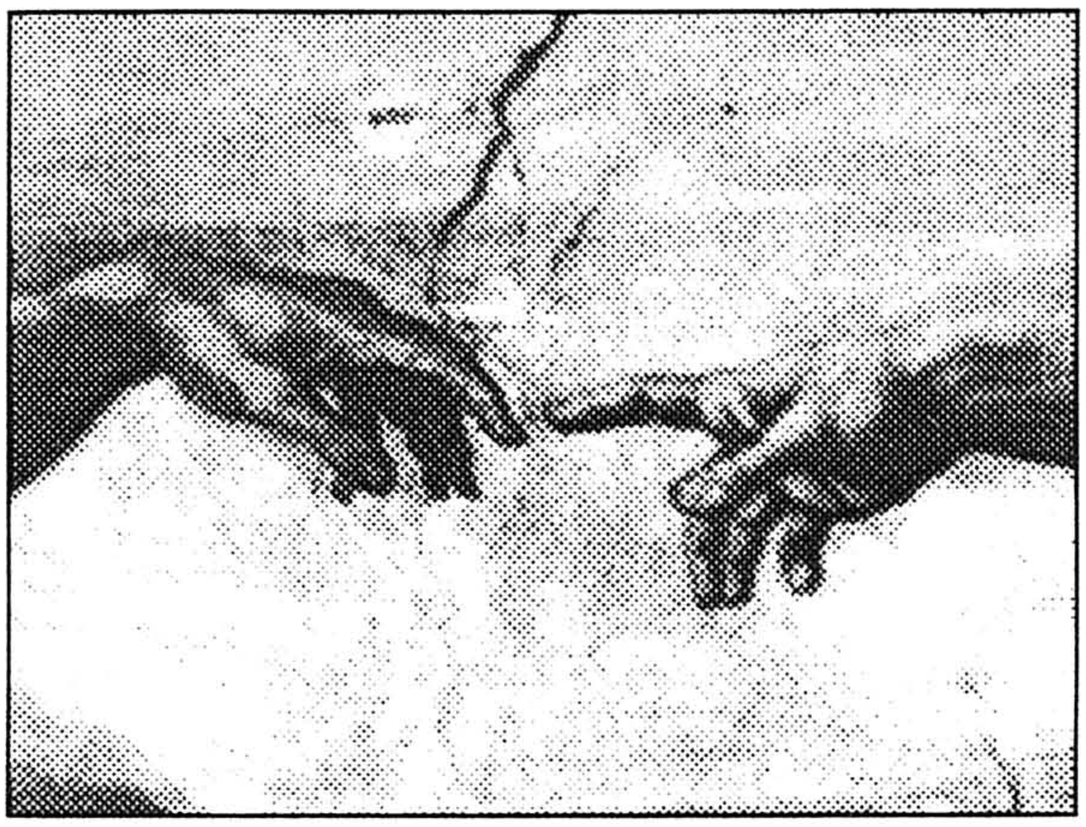


Fig. 18 - Vessel tracking



CAPÍTULO 3

ALGORITMO DESENVOLVIDO

Neste capítulo iremos descrever detalhadamente o algoritmo de processamento e análise das imagens histeroscópicas, e que inclui um método alternativo de detecção de redes vasculares.

3.1 Descrição

No sentido de cumprir o objectivo apresentado no primeiro capítulo: caracterizar objectiva e quantitativamente as redes vasculares uterinas existentes nas imagens histeroscópicas; foi desenvolvido um algoritmo de processamento e análise dessas mesmas imagens. Como se ilustra na Fig. 19 este algoritmo é composto pelas seguintes três fases:

- pré-processamento
- processamento
- análise

No pré-processamento pretende-se melhorar algumas das características apresentadas pelas imagens, nomeadamente o seu contraste. Na fase de processamento irá ser efectuada a operação de detecção da rede vascular. Por último, na fase de análise são extraídas as características que definem a rede.

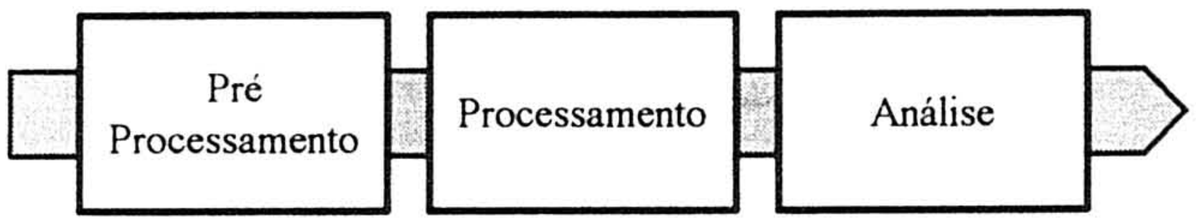


Fig. 19 - Fases do algoritmo desenvolvido

Nas secções seguintes serão descritas cada uma destas fases assim como as operações que as compõem.

3.2 Pré-processamento

A fase de pré-processamento destina-se a melhorar as características das imagens facilitando assim a fase de processamento seguinte. Para tal recorre a duas operações:

- remoção da superfície base
- equalização do histograma

Na primeira das operações pretende-se eliminar a variabilidade de intensidade do fundo das imagens. A equalização do histograma provoca, por seu lado, um aumento do contraste apresentado por essas mesmas imagens.

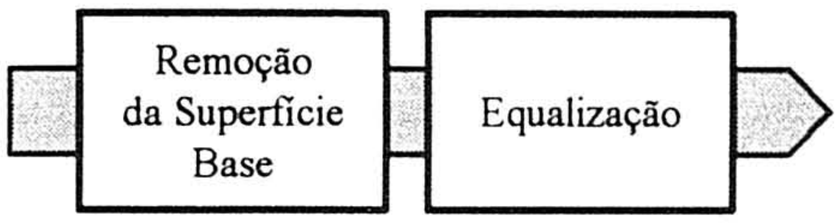


Fig. 20 - Pré-processamento

3.2.1 Remoção da superfície base

A intensidade do fundo das imagens não é constante. Com efeito, existe como que uma superfície base de intensidades. Para este facto contribui a forma do útero que não é plana mas sim em forma de sela, bem como o tipo de iluminação usado que sendo muito direccional torna as zonas periféricas da imagem progressivamente mais escuras.

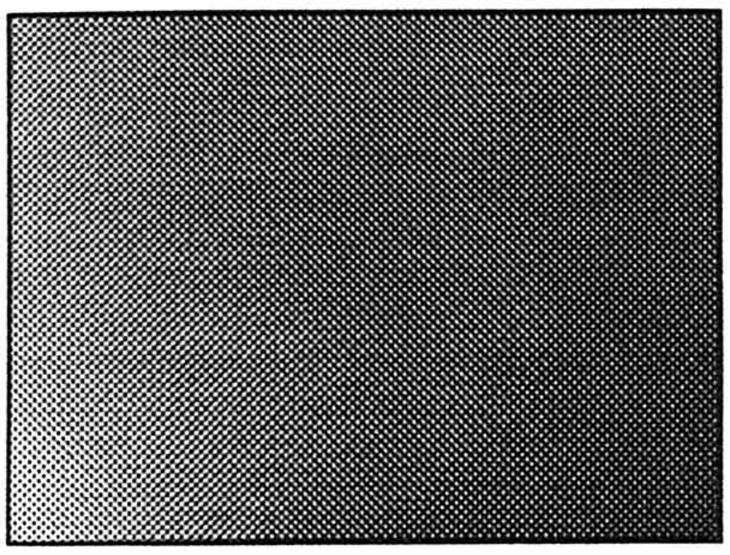


Fig. 21 - Superfície base

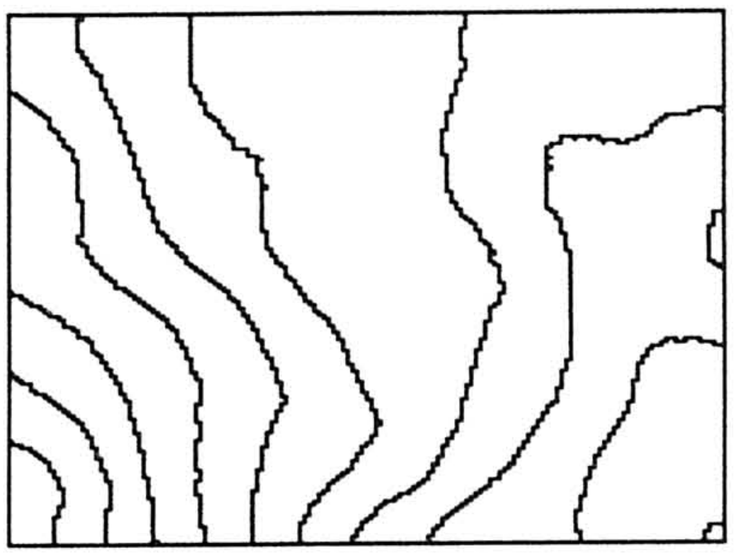


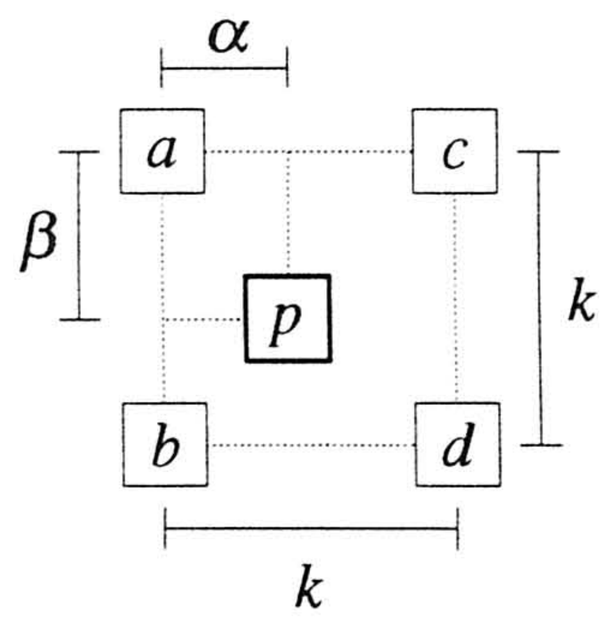
Fig. 22 - Curvas de intensidade

Este facto apresenta dois inconvenientes. Por um lado impede uma correcta avaliação da intensidade de vasos situados em diferentes zonas da imagem, por outro alarga de modo artificial a gama de intensidades impedindo assim um maior aumento de contraste por parte da operação de equalização.

Nesta operação a imagem final será igual à inicial subtraída da superfície base. Para que os resultados não sejam negativos teremos que somar a intensidade média da superfície.

$$out = in - surface + \mu_{surface}$$

Para se obter a desejada superfície base recorre-se a um método já utilizado com sucesso no processamento de sinais impedo-cardiográficos. Neste método a imagem é decimada por um factor *k*. Na nova imagem, possuidora de um menor número de pontos, cada ponto representa a intensidade média duma janela de *k* por *k* pontos da imagem original. Seguidamente é estimada a superfície após o que se interpola a imagem com um factor *k* idêntico ao utilizado na decimação. A interpolação utilizada é a bilinear onde:



$$p = \frac{(k-\alpha).(k-\beta).a + (k-\alpha).\beta.b + \alpha.(k-\beta).c + \alpha.\beta.d}{k^2}$$

A estimação da superfície poderá ser efectuada de duas formas: por aproximação duma superfície matemática previamente definida ou por filtragem.

A. Aproximação por uma superfície matemática

Verifica-se que dada a geometria e condições de iluminação do orgão em estudo pode-se aproximar a superfície base dentro da região de interesse a um parabolóide.

$$z_e = ax^2 + by^2 + cx + dy + exy + f$$

Utilizando um critério de minimização do erro quadrático médio é possível, como veremos seguidamente, determinar o valor dos parâmetros a, b, c, d, e e f desse parabolóide.

Defina-se E como o erro quadrático médio (somatório das diferenças entre o quadrado do valor real z_r e o quadrado do valor estimado z_e).

$$E = \sum \left({z_r}^2 - {z_e}^2 \right) = \sum \left({z_r}^2 - \left(ax^2 + by^2 + cx + dy + exy + f \right)^2 \right)$$

Para minimizar E é necessário tornar nulas as derivadas em ordem a cada um dos parâmetros.

$$\begin{cases} \frac{\partial E}{\partial a} = 0 \\ \frac{\partial E}{\partial b} = 0 \\ \frac{\partial E}{\partial c} = 0 \\ \frac{\partial E}{\partial d} = 0 \\ \frac{\partial E}{\partial e} = 0 \\ \frac{\partial E}{\partial f} = 0 \end{cases}$$

$$\begin{cases} a\sum x^4y^0 + b\sum x^2y^2 + c\sum x^3y^0 + d\sum x^2y^1 + e\sum x^3y^1 + f\sum x^2y^0 = \sum x^2y^0 z_r \\ a\sum x^2y^2 + b\sum x^0y^4 + c\sum x^1y^2 + d\sum x^0y^3 + e\sum x^1y^3 + f\sum x^0y^2 = \sum x^0y^2 z_r \\ a\sum x^3y^0 + b\sum x^1y^2 + c\sum x^2y^0 + d\sum x^1y^1 + e\sum x^2y^1 + f\sum x^1y^0 = \sum x^1y^0 z_r \\ a\sum x^2y^1 + b\sum x^0y^3 + c\sum x^1y^1 + d\sum x^0y^2 + e\sum x^1y^2 + f\sum x^0y^1 = \sum x^0y^1 z_r \\ a\sum x^3y^1 + b\sum x^1y^3 + c\sum x^2y^1 + d\sum x^1y^2 + e\sum x^2y^2 + f\sum x^1y^1 = \sum x^1y^1 z_r \\ a\sum x^2y^0 + b\sum x^0y^2 + c\sum x^1y^0 + d\sum x^0y^1 + e\sum x^1y^1 + f\sum x^0y^0 = \sum x^0y^0 z_r \end{cases}$$

A resolução deste sistema de seis equações a seis incógnitas permite determinar o valor dos parâmetros do parabolóide.

Note-se ainda que os somatórios presentes no primeiro membro das equações são de cálculo directo, bastando para isso aplicar as seguintes fórmulas:

$$\sum x^m y^n = \sum_{x=0}^{N_x} \sum_{y=0}^{N_y} x^m y^n = \sum_{x=0}^{N_x} x^m \sum_{y=0}^{N_y} y^n$$

$$\sum_{x=0}^{N} x^0 = N+1$$

$$\sum_{x=0}^{N} x^1 = \frac{N(N+1)}{2} \qquad \sum_{x=0}^{N} x^3 = \left(\sum_{x=0}^{N} x^1 \right)^2$$

$$\sum_{x=0}^{N} x^2 = \frac{2N+1}{3} \sum_{x=0}^{N} x^1 \qquad \sum_{x=0}^{N} x^4 = \frac{3N^2+3N-1}{5} \sum_{x=0}^{N} x^2$$

B. Filtragem

Uma outra forma de estimar a superfície base consiste na filtragem da imagem decimada com um filtro passa-baixo. Este filtro deverá idealmente eliminar as componentes de alta frequência correspondentes aos vasos, e deixar inalteradas as componentes de muito baixa frequência a que corresponde a superfície base.

Este método é mais simples e rápido que o anterior, apresentando a vantagem adicional de ser genérico uma vez que é independente da forma da superfície base. Contudo, uma vez que os filtros utilizados não são ideais, a superfície estimada irá ainda conter componentes de alta frequência embora bastante atenuadas. Este facto provoca alguma atenuação dos vasos na imagem final. Esta atenuação não é no entanto muito significativa, como se pode constatar nas figuras seguintes.

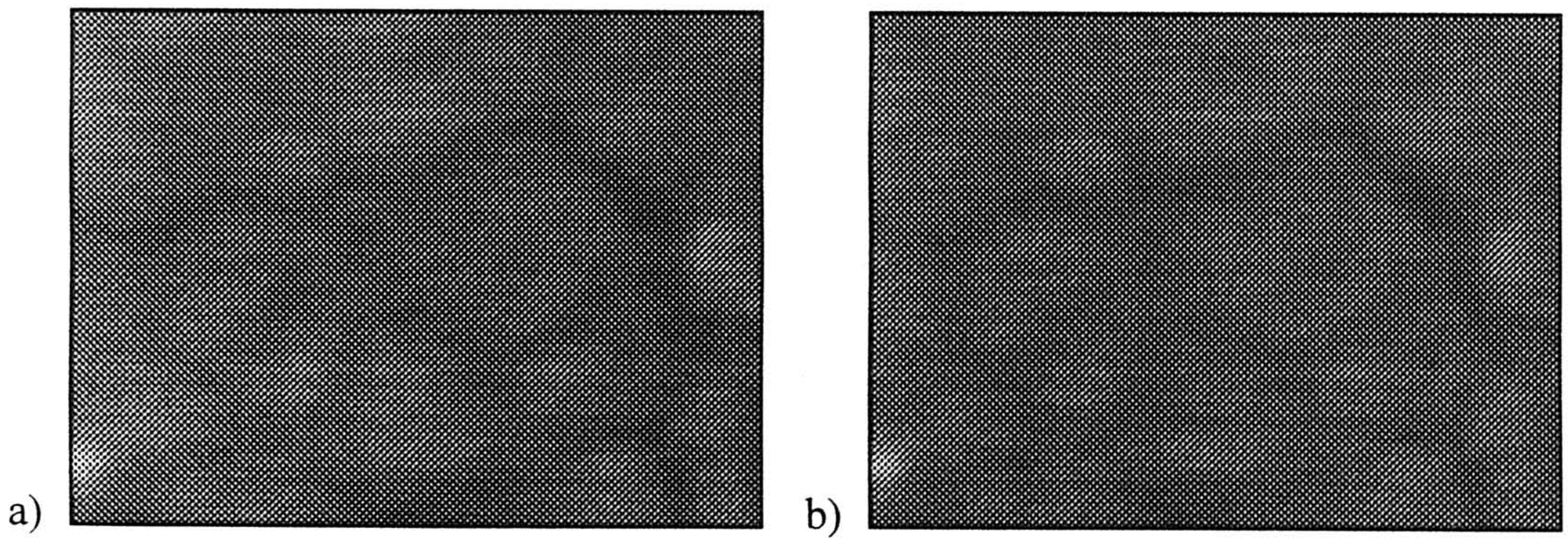


Fig. 23 - Remoção da superfície base: a) Método A, b) Método B

Foi este último método o escolhido para estimar a superfície base. Na prática são efectuadas duas filtragens consecutivas utilizando um filtro de média de dimensão 5. O factor *k* usado quer na decimação quer na interpolação é também igual a 5.

3.2.2 Equalização do histograma

Com a equalização do histograma pretende-se aumentar o contraste das imagens uma vez que este é muito reduzido. O baixo contraste perfeitamente notório em termos visuais é confirmado pela concentração da grande maioria dos pixels da imagem numa pequena gama de intensidades do histograma.

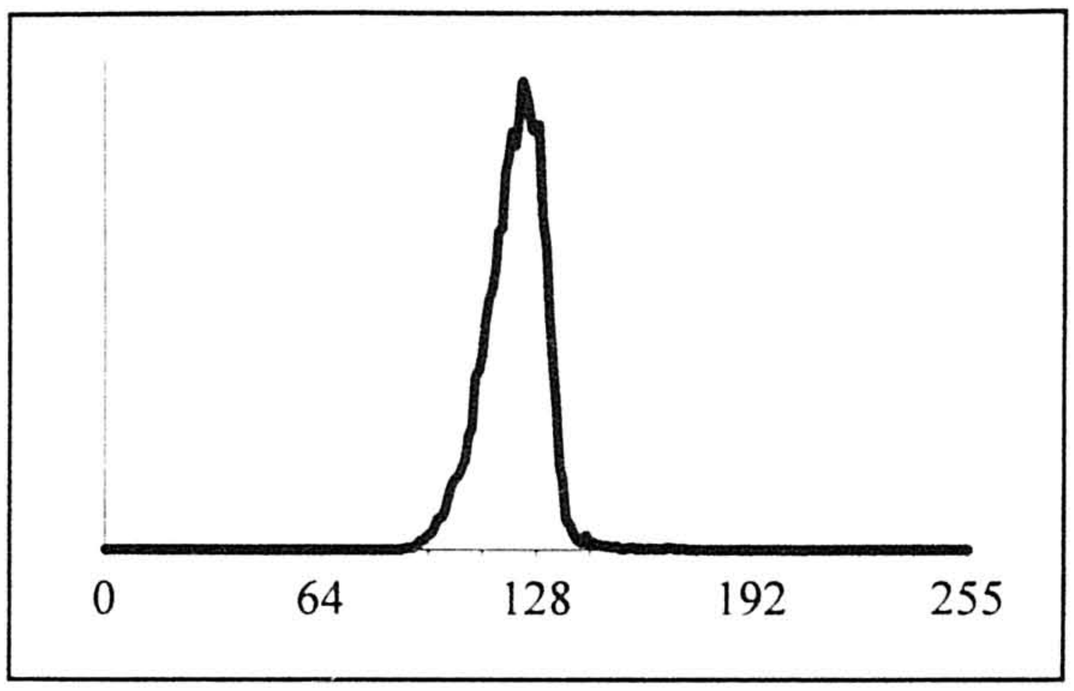


Fig. 24 - Histograma

A operação de equalização não faz mais do que alargar a gama de intensidades provocando assim o desejado aumento de contraste. Existem basicamente dois processos para se efectuar este alargamento: de forma linear e de forma não linear.

A equalização não linear afasta as riscas do histograma proporcionalmente à sua amplitude. Isto é, riscas correspondentes a intensidades com um grande número de pixels serão mais afastadas do que riscas de menor peso.

Pelo contrário, na equalização linear as riscas são afastadas de igual forma. Este facto embora possa conduzir em certos casos a um menor aumento de contraste evita contudo situações de eliminação abusiva de determinados detalhes na imagem motivada pela fusão de diferentes riscas numa só.

Consequentemente a equalização escolhida foi a linear. Note-se que neste tipo de equalização o aumento de contraste é tanto maior quanto mais concentrado se encontrar o histograma, daí que a remoção da superfície base auxilie também neste processo ao concentrar algo mais o histograma. O resultado desta operação pode ser visto na Fig. 25.

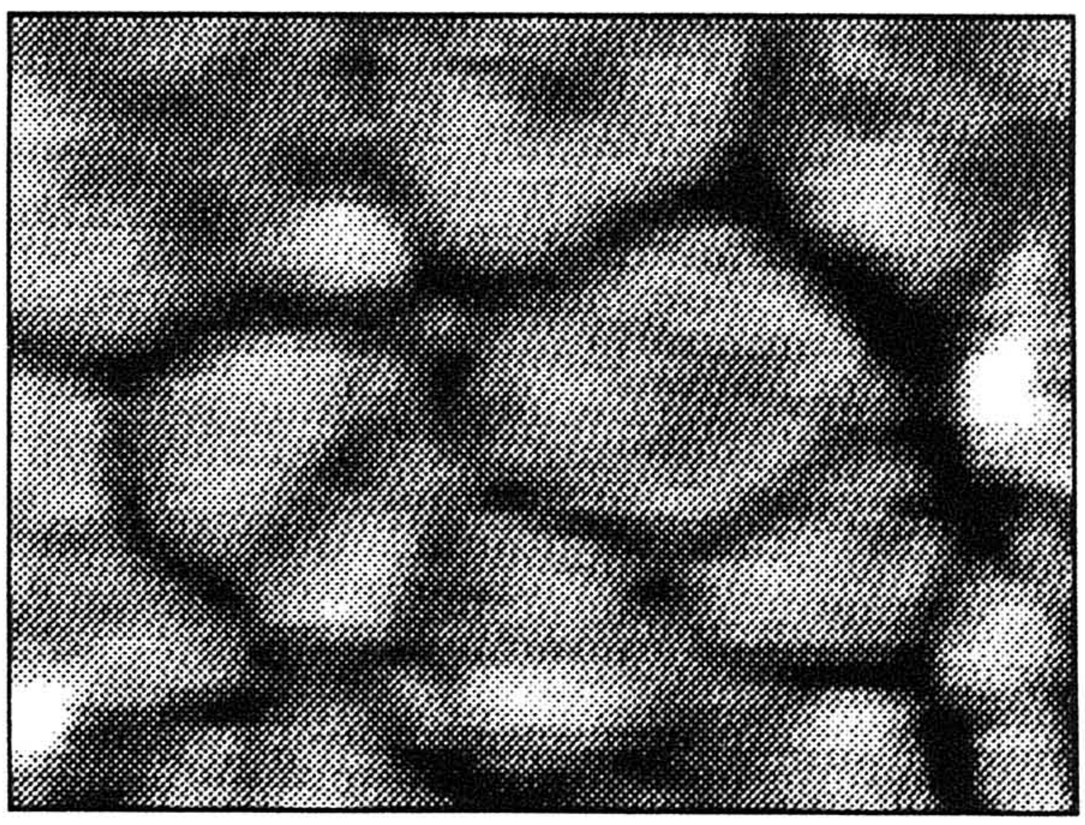


Fig. 25 - Imagem equalizada

3.3 Processamento

É na fase de processamento que se efectua a detecção da rede vascular uterina. Como se constatou no capítulo 2 nenhum dos métodos de detecção aí descritos apresenta bons resultados dadas as características adversas das imagens histeroscópicas. Não obstante o aumento de contraste obtido pelas operações de pré-processamento as imagens continuam a apresentar vasos de diferentes intensidades, contornos mal definidos e cuja característica de continuidade é posta em causa devido ao seu desaparecimento sob a mucosa. Todos estes factos obrigaram ao desenvolvimento de um método alternativo para efectuar a desejada detecção.

Este método explora todas as características das redes vasculares (*C1*, *C2* e *C3*), tendo uma abordagem simultaneamente local e global o que permite a obtenção de bons resultados mesmo quando as imagens possuem pouco contraste e as redes encontram-se muito esbatidas. O método de detecção desenvolvido é composto pelas seguintes cinco operações:

- detecção de mínimos locais
- thinning
- decomposição em segmentos
- classificação dos segmentos
- reconstrução da rede vascular

Em primeiro lugar é obtida uma imagem de mínimos locais de intensidade com base numa medida de contraste totalmente inovadora. Esta imagem é depois simplificada por intermédio duma operação de thinning. O esqueleto assim obtido será decomposto em vários segmentos que são seguidamente classificados. Por último será efectuada a reconstrução da rede vascular a partir dos segmentos classificados como vasos.

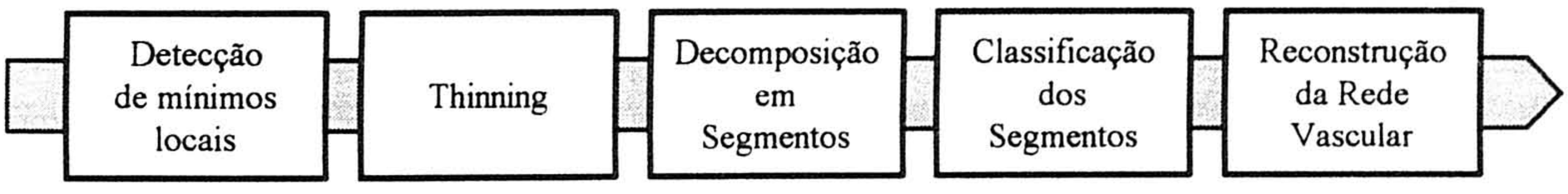
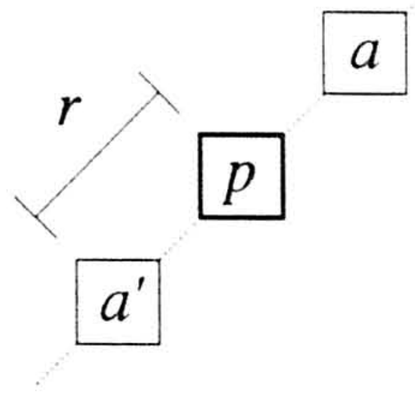


Fig. 26 - Processamento

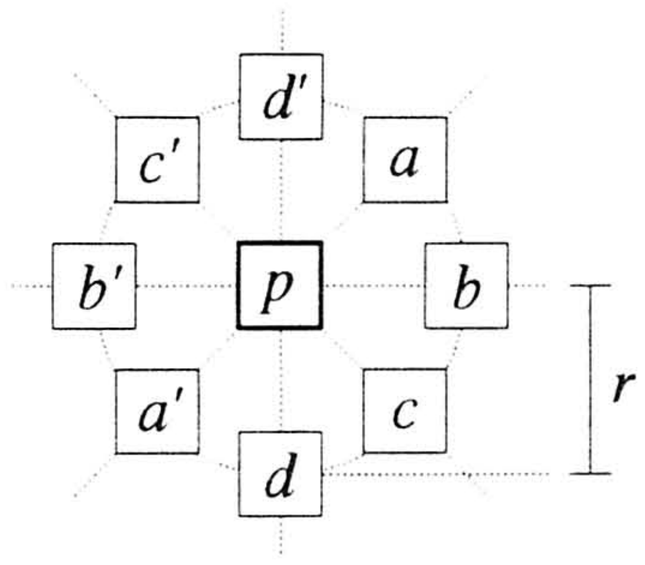
3.3.1 Detecção de mínimos locais

Um pixel é mínimo local segundo uma dada direcção se a sua intensidade for menor que as intensidades de ambos os pixels situados ao seu lado a uma distância r. A menor das duas diferenças de intensidade é designada por contraste direccional.



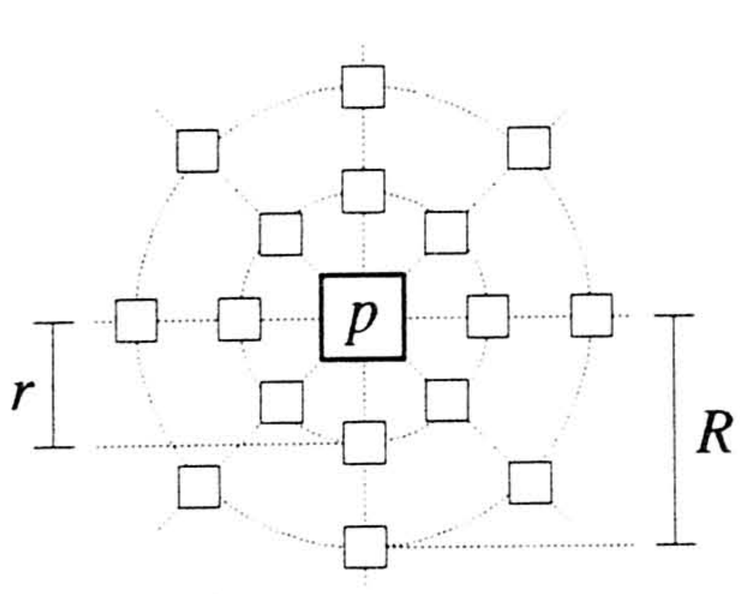
$$c_{aa'} = \min(a - p, a' - p)$$

Para que um pixel seja mínimo local basta que o seja em pelo menos uma das quatro direcções: horizontal, vertical e as duas diagonais. O contraste será o maior dos contrastes direccionais e permite quantificar o quanto um pixel é mínimo local.



$$c_r = \max(c_{aa'}, c_{bb'}, c_{cc'}, c_{dd'})$$

O raio da coroa de pixels em redor do pixel em análise (distância r) condiciona o resultado obtido. Distâncias pequenas não permitem detectar vasos de grande calibre pois toda a coroa de pixels fica dentro do vaso. A utilização de grandes distâncias poderá provocar a perda de vasos de pequeno calibre. Isto conduziu à utilização dum processo misto em que se utilizam duas distâncias. O contraste final será o maior dos contrastes obtidos em cada uma das coroas.



$$c = \max(c_r, c_R)$$

Por forma a que a imagem de contraste assim obtida não seja excessivamente ruidosa deverá ser efectuada uma filtragem passa-baixo antes e depois da sua determinação. A imagem de contraste pode ser vista na Fig. 27 tendo sido utilizados os valores 3 e 6 para as distâncias *r* e *R* respectivamente

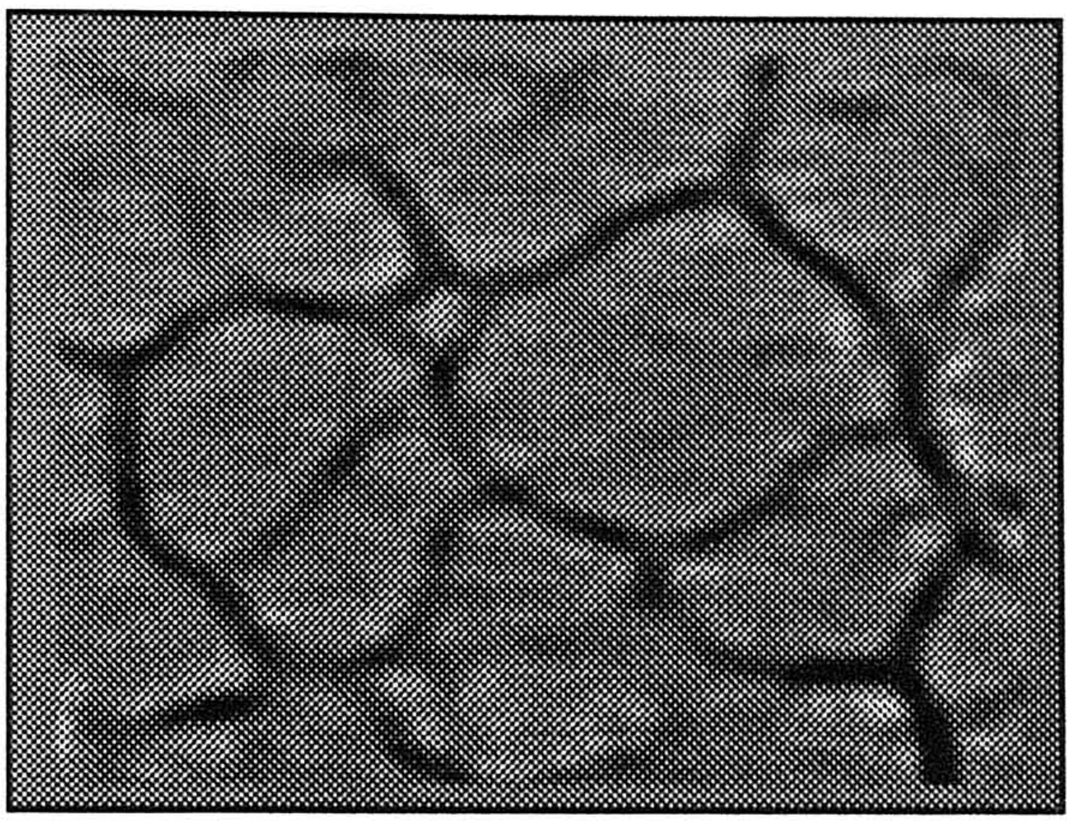


Fig. 27 - Imagem de contraste

A imagem de contraste será seguidamente binarizada escolhendo-se para tal um nível de contraste mínimo admissível acima do qual os pixels são considerados efectivamente mínimos locais. Este nível *mpc* (minimum pixel contrast) não é determinado automaticamente dada a grande variabilidade de contraste apresentado pelas imagens. O operador do sistema deverá pois seleccionar o nível que conduza a uma imagem de mínimos locais que melhor represente toda a estrutura de rede vascular.

As Figs. 28a e 28b mostram o resultado da binarização da imagem de contraste usando um nível *mpc* igual a 1 e 36 respectivamente. De notar que a primeira imagem de mínimos locais ao contrário da segunda contém a totalidade da estrutura da rede não obstante possuir muitos pixels pertencentes ao fundo da imagem. Este último facto não constitui problema devido às operações subsequentes.

a) b)

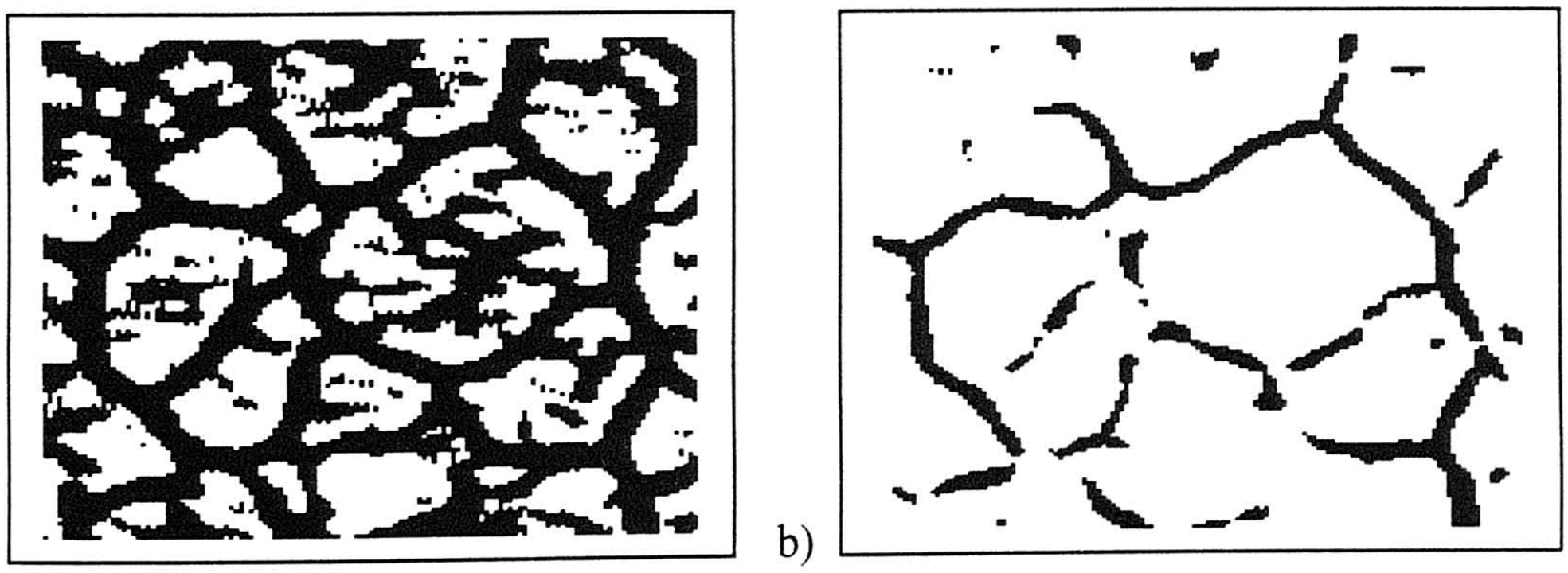


Fig. 28 - Mínimos locais: a) *mpc* = 1, b) *mpc* = 36

3.3.2 Thinning

Uma vez obtida a imagem de mínimos locais e no intuito de a simplificar sem contudo alterar a sua conectividade, é executada uma operação de thinning.

No método usado, denominado NSEW, os pontos fronteiros segundo uma dada direcção são retirados se isso não violar as condições de conectividade da imagem. Por forma a que o esqueleto seja o mais centrado possível este procedimento deverá ser executado alternadamente segundo as quatro direcções: Norte, Sul, Este e Oeste.

Assumindo que os pontos de fundo têm valor 0 e os restantes valor 1 e que *n*, *ne*, *e*, *se*, *s*, *sw*, *w*, e *nw* representam os pontos vizinhos do ponto *p* em análise, é possível efectuar as seguintes definições:

o ponto p é fronteiro segundo a direcção x sse:

$$x = 0$$
$$p = 1$$

a retirada do ponto p não viola as condições de conectividade sse:

$$n + ne + e + se + s + sw + w + nw \neq 1$$
$$\overline{n}.ne.\overline{e} = 0$$
$$\overline{e}.se.\overline{s} = 0$$
$$\overline{s}.sw.\overline{w} = 0$$
$$\overline{w}.nw.\overline{n} = 0$$
$$\overline{n}.e.\overline{s}.w = 0$$
$$n.\overline{e}.s.\overline{w} = 0$$

A Fig. 29 mostra o esqueleto da imagem de mínimos locais anteriormente determinada.

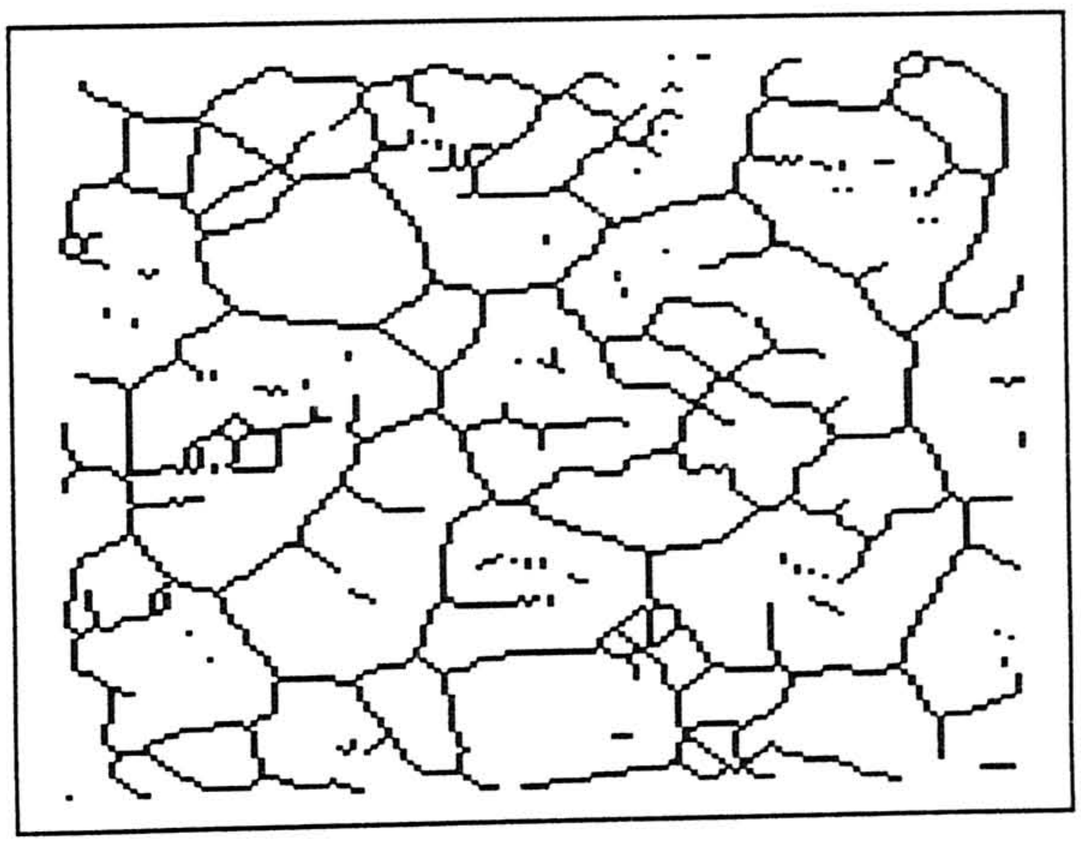
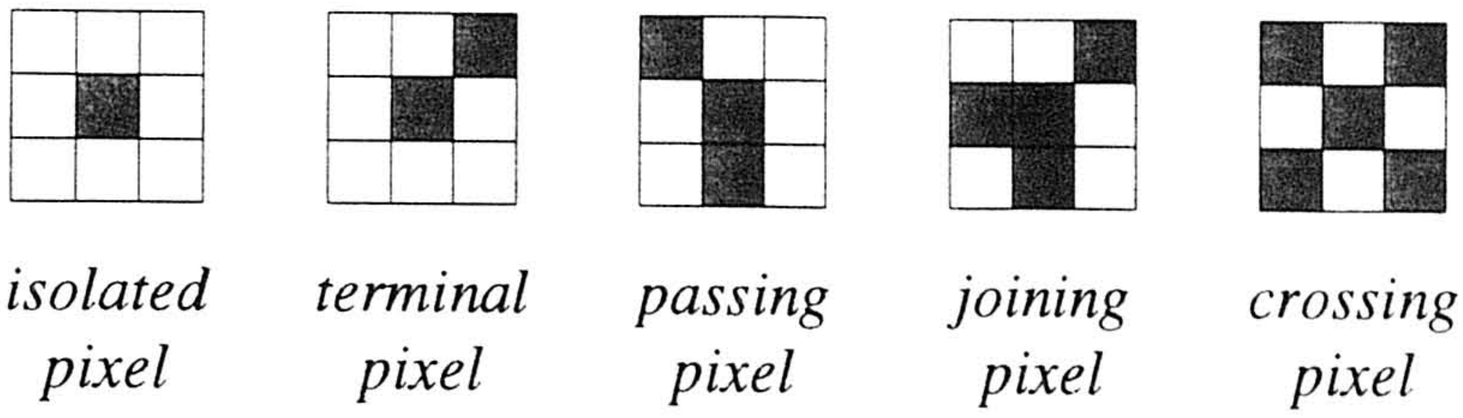


Fig. 29 - Esqueleto

3.3.3 Decomposição em segmentos

Os pixels do esqueleto têm diferentes características em função do número de pixels vizinhos activos. Um pixel será isolado, terminal, de passagem, entroncamento ou cruzamento consoante esse número seja respectivamente igual a 0, 1, 2, 3 e 4. A operação de thinning garante a não existência de situações em que este número é igual ou superior a 5.



isolated pixel *terminal pixel* *passing pixel* *joining pixel* *crossing pixel*

Um processo simples e rápido que permite efectuar a desejada decomposição do esqueleto em segmentos é a retirada de todos os pixels que não sejam terminais ou de passagem.

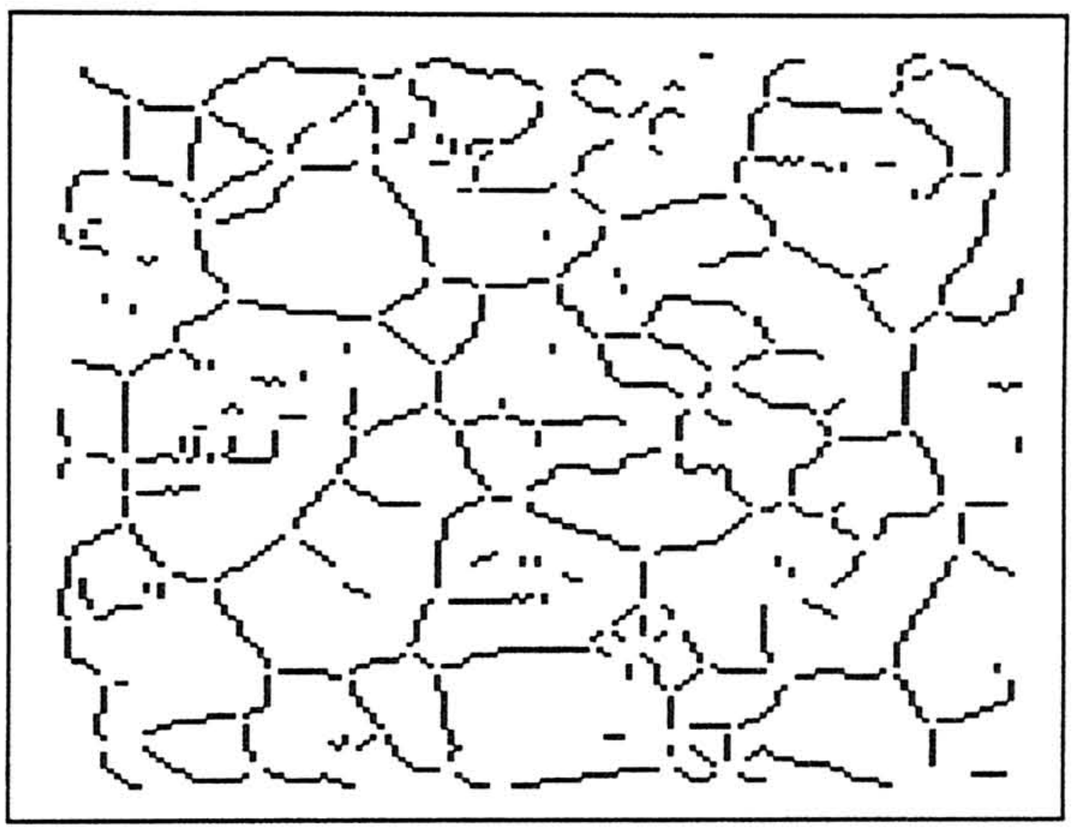


Fig. 30 - Esqueleto decomposto em segmentos

3.3.4 Classificação dos segmentos

A fase de classificação dos segmentos visa a escolha daqueles que são pelas suas características mais susceptíveis de constituírem vasos. Estas características são:

- a intensidade média
- o contraste médio

A intensidade e o contraste médio são obtidos a partir da informação contida respectivamente na imagem equalizada e de contraste anteriormente determinadas.

O classificador utilizado é extremamente simples e estipula que para um segmento ser considerado vaso terá que se verificar a seguinte condição:

$$(Contrast > C) \vee (Intensity < I)$$

O valor utilizado para o parâmetro *I* foi fixo após vários testes em 60. O parâmetro *C* também designado por *msc* (minimum segment contrast) deverá ser seleccionado pelo operador do sistema devido à já mencionada grande variabilidade de contraste apresentada pelas imagens. A selecção deste nível deverá permitir a eliminação de todos os segmentos não pertencentes à rede vascular.

As Figs. 31 e 32 mostram quais os segmentos classificados como vasos quando o nível *msc* é seleccionado de forma incorrecta e correcta respectivamente.

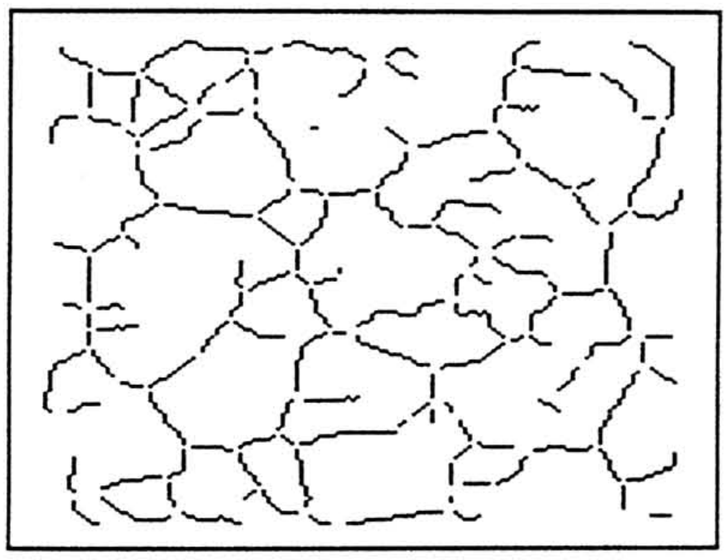
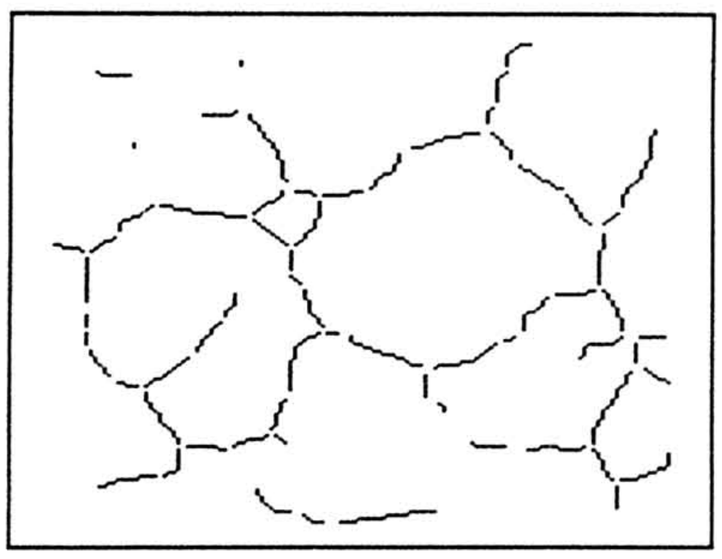
a)  b) 

Fig. 31 - Classificação incorrecta: a) *msc* = 8, b) *msc* = 30

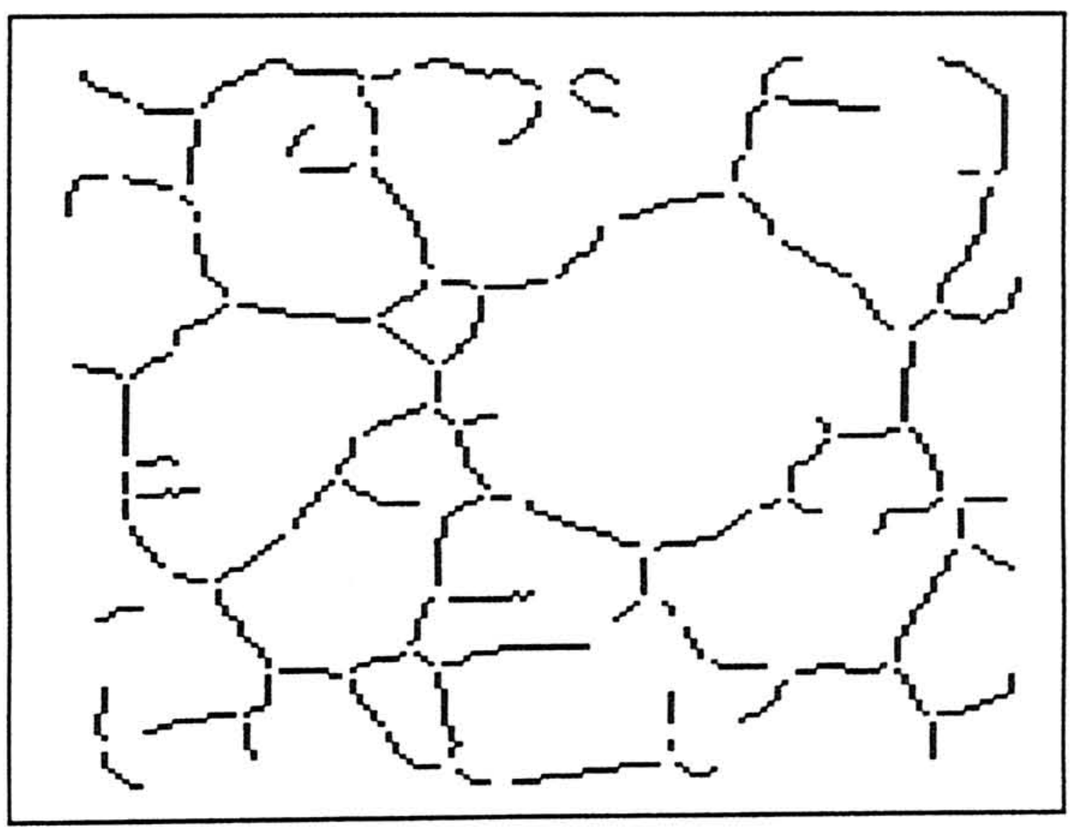


Fig. 32 - Segmentos classificados correctamente como vasos: *msc* = 13

3.3.5 Reconstrução da rede vascular

Um vez conhecida a localização dos vasos há que reconstruir a rede vascular. Para tal, em todos os pixels dos segmentos classificados como vasos será desenhado um círculo de diâmetro igual ao calibre do vaso nesse ponto. O desenho sucessivo desses círculos dará origem à rede vascular. Por forma a suavizar os seus contornos é efectuada uma operação de erosão seguida de dilatação.

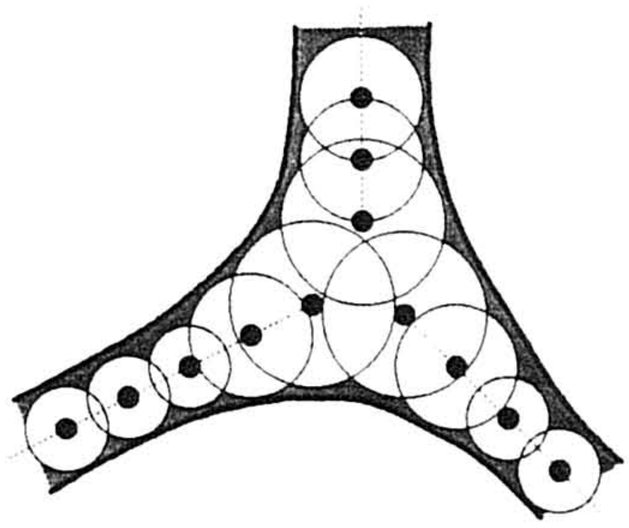


Fig. 33 - Reconstrução da rede vascular

A determinação do calibre é realizada por análise do perfil de intensidades obtido na imagem equalizada segundo a direcção de maior contraste. Sendo o calibre igual à distância entre as arestas do vaso teremos que determinar a localização destas. Vários critérios podem ser utilizados para esta localização: pontos de máxima derivada ou simplesmente pontos de passagem pela intensidade *m* do perfil, onde:

$$m = \frac{\max_{int} + \min_{int}}{2}$$

A simplicidade deste último método de localização de arestas aliada aos seus bons resultados justificou a sua escolha. Na imagem seguinte é apresentada a rede vascular assim reconstruída.

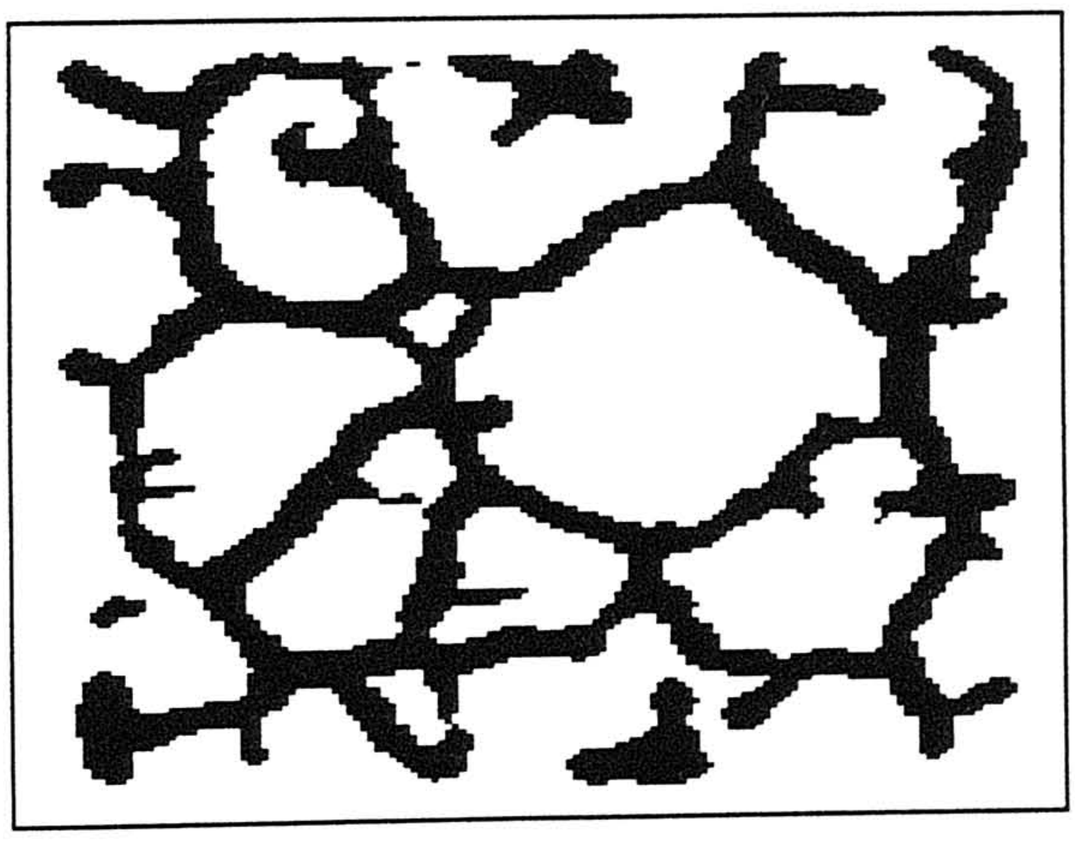


Fig. 34 - Imagem final da rede vascular

3.4 Análise

A última fase deste algoritmo procede à análise da rede vascular que foi entretanto obtida. Esta fase é composta por uma única operação designada por:

- extracção de características

Esta operação irá dar como resultado final um conjunto de características que permitem a desejada caracterização precisa, objectiva e quantitativa das redes vasculares uterinas.

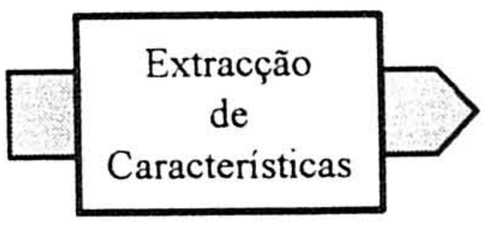


Fig. 35 - Análise

3.4.1 Extracção de características

Após a obtenção de uma imagem perfeitamente segmentada da rede vascular é agora possível extrair de uma forma relativamente simples um conjunto de características capazes de a definirem objectiva e quantitativamente. Estas características foram encontradas conciliando as necessidades clínicas apontadas pelo Dr. Silva Carvalho com a exequibilidade da sua determinação em tempo útil. Deste compromisso resultaram as seguintes características:

F1	área vascular
F2	perímetro vascular
F3	número de segmentos
F4	número de nodos
F5	calibre mínimo
F6	calibre médio
F7	calibre máximo
F8	comprimento mínimo
F9	comprimento médio
F10	comprimento máximo
F11	homogeneidade da área
F12	homogeneidade do perímetro
F13	homogeneidade do calibre
F14	homogeneidade dos nodos

Estas características são avaliadas por análise não só da imagem final da rede vascular reconstruída mas também de algumas imagens obtidas ao longo do processamento, nomeadamente a imagem de segmentos classificados como vasos. Seguidamente será descrita de forma sucinta o meio de determinação destas mesmas características.

Assim a área vascular não é mais do que o número de pixels pretos na imagem final. De notar que as quatro primeiras características são fornecidas em termos percentuais da região de interesse dada a variabilidade de dimensões que esta pode apresentar. O perímetro corresponde ao número de pixels pertencentes às arestas da rede. Para tal a imagem final é operada com um filtro detector de arestas (Laplaciano). Apesar da simplicidade deste filtro o resultado é perfeito uma vez que a imagem a operar é binária.

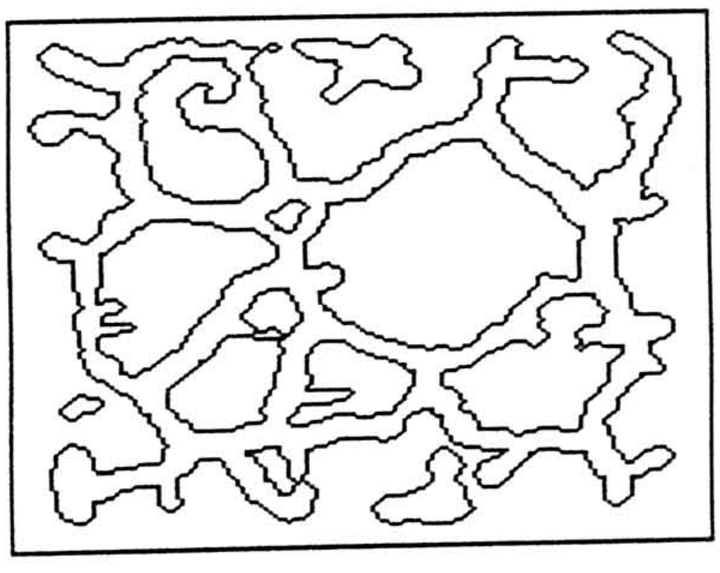


Fig. 36 - Contornos da rede

O número de segmentos e nodos é avaliado na imagem de segmentos classificados como vasos, cuja estrutura facilita a sua determinação. Cada conjunto contíguo de pixels define um segmento. Um nodo corresponde a um local de convergência de três ou mais segmentos. Ambas estas características fornecem uma informação sobre a complexidade e forma de estruturação da rede. O calibre (distância entre as arestas do vaso) é obtido durante a operação de reconstrução da rede e está directamente ligado ao fluxo sanguíneo que atravessa o vaso. Os comprimentos são também obtidos na imagem de segmentos classificados como vasos. É de realçar o facto de o comprimento médio ser obtido através de uma média pesada. Por fim as homogeneidades procuram representar a maior ou menor uniformidade na distribuição de uma determinada característica pela ROI. Para tal a ROI é dividida numa grelha de quadrados de 10 por 10 pixels, após o que é determinado o valor médio tomado pela característica em análise em cada um deles. Estes valores constituem como que uma função *f(x,y)* cujos desvios padrão segundo x e segundo y são:

$$\mu_x = \frac{\sum x.f(x,y)}{\sum f(x,y)} \qquad \mu_y = \frac{\sum y.f(x,y)}{\sum f(x,y)}$$

$$\sigma_x = \sqrt{\frac{\sum (\mu_x - x)^2 f(x,y)}{\sum f(x,y)}} \qquad \sigma_y = \sqrt{\frac{\sum (\mu_y - y)^2 f(x,y)}{\sum f(x,y)}}$$

Estes valores deverão ser normalizados uma vez que as dimensões da ROI não são fixas.

$$\sigma_x^N = \frac{\sigma_x}{N_x} \qquad \sigma_y^N = \frac{\sigma_y}{N_y}$$

A homogeneidade é finalmente obtida efectuando a média geométrica dos dois valores.

$$Homogeneidade = \sqrt{\sigma_x^N \sigma_y^N}$$

Seguidamente será apresentada a lista de características e respectivos valores resultantes da análise da rede vascular contida na ROI que tem vindo a servir de exemplo ao longo deste capítulo.

Área vascular	***29.40306***	*%*
Perímetro vascular	***9.62755***	*%*
Número de segmentos	***0.54082***	*%*
Número de nodos	***0.53571***	*%*
Calibre mínimo	***1***	*p*
Calibre médio	***4***	*p*
Calibre máximo	***7***	*p*
Comprimento mínimo	***2***	*p*
Comprimento médio	***9***	*p*
Comprimento máximo	***24***	*p*
Homogeneidade de área	***1.83611***	
Homogeneidade de perímetro	***1.05680***	
Homogeneidade de calibre	***1.06918***	
Homogeneidade de perímetro	***1.06484***	



CAPÍTULO 4

SISTEMA

Neste capítulo iremos apresentar o programa SPIMUT que implementa o algoritmo anteriormente descrito e assim como o sistema de hardware onde ele se integra.

4.1 Hardware

O hardware do sistema é composto por um computador pessoal ao qual está ligada uma placa de aquisição de imagem. Na entrada desta placa é ligado um videogravador encarregue da reprodução das imagens histeroscópicas previamente gravadas. Na saída da placa existe um monitor auxiliar para visualização dessas mesmas imagens.

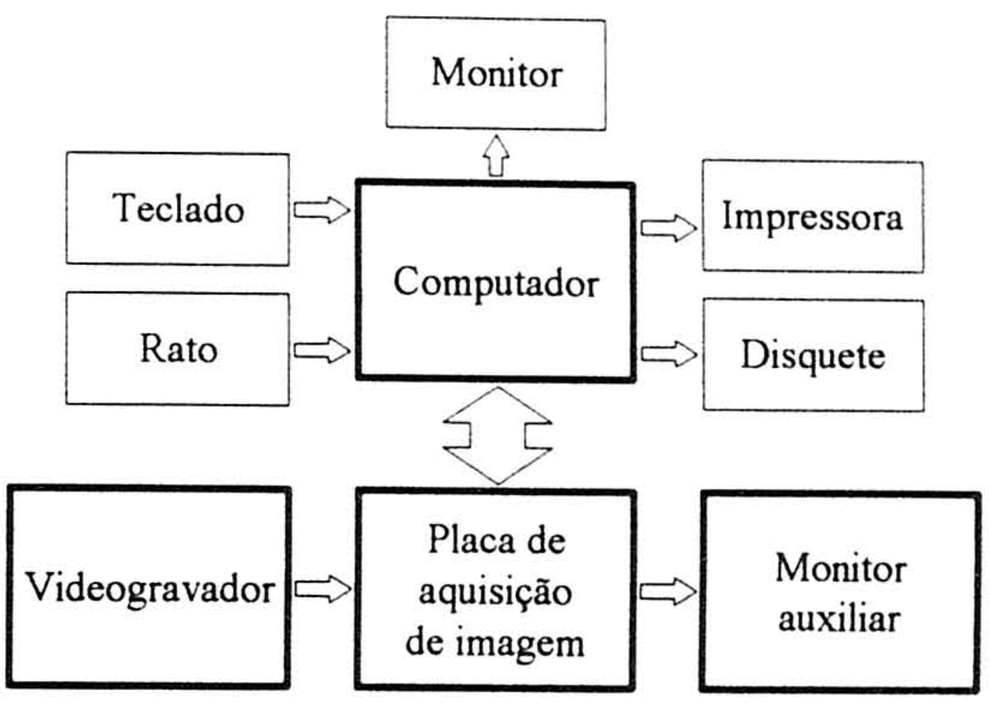


Fig. 37 - Diagrama de blocos do sistema

A placa de aquisição de imagem (IRIS DT2851) da Data Translation digitaliza os sinais de vídeo em imagens monocromáticas de 512 por 512 pontos. Esta placa pode ainda efectuar operações muito simples sobre as imagens recorrendo ao seu processador de LUTs. Tal como pode ser visto no diagrama de blocos apresentado na Fig. 38, a DT2851 possui:

- 8 LUTs de entrada (256 x 8 bit)
- 2 buffers de memória (512 x 512 x 8 bit)
- 8 LUTs de saída (256 x 24 bit)
- 2 portos de vídeo externos
- 3 saídas RGB (8 bit)

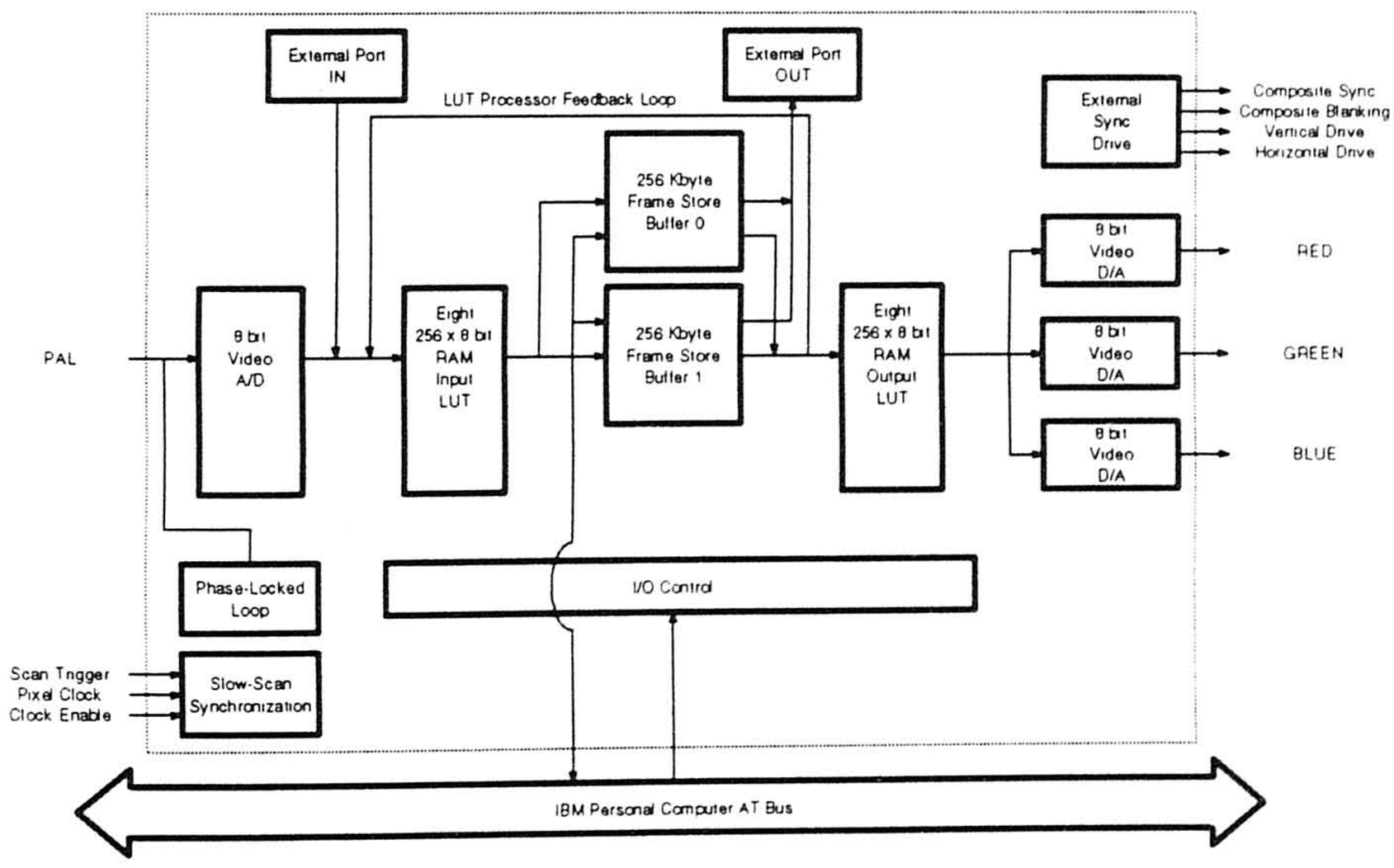


Fig. 38 - Diagrama de blocos da placa DT2851

4.2 Software

O programa SPIMUT (Sistema de Processamento de Imagens Histeroscópicas Uterinas) foi desenvolvido em linguagem C recorrendo ao compilador da Microsoft versão 7.0 por questões de compatibilidade com a biblioteca de funções (iscllib.lib) que acompanha a placa de aquisição de imagem. Este programa executa todas as operações de processamento e análise descritas no capítulo anterior para além das tarefas de aquisição, leitura e armazenamento das imagens, selecção da região de interesse e ainda criação de ficheiros de resultados para posterior análise estatística.

A principal preocupação tida durante o desenvolvimento deste produto foi a sua simplicidade de utilização, uma vez que este se destina a pessoal não especializado no campo informático. Para tal o programa é fortemente interactivo com a utilização de menus e botões aos quais estão associadas as acções a realizar. Acresce a este facto o uso do rato como elemento preferencial na entrada de dados sendo o teclado reservado para as operações de introdução de nomes de ficheiros. A utilização do programa SPIMUT é pois bastante intuitiva como pode ser visto na Fig. 39.

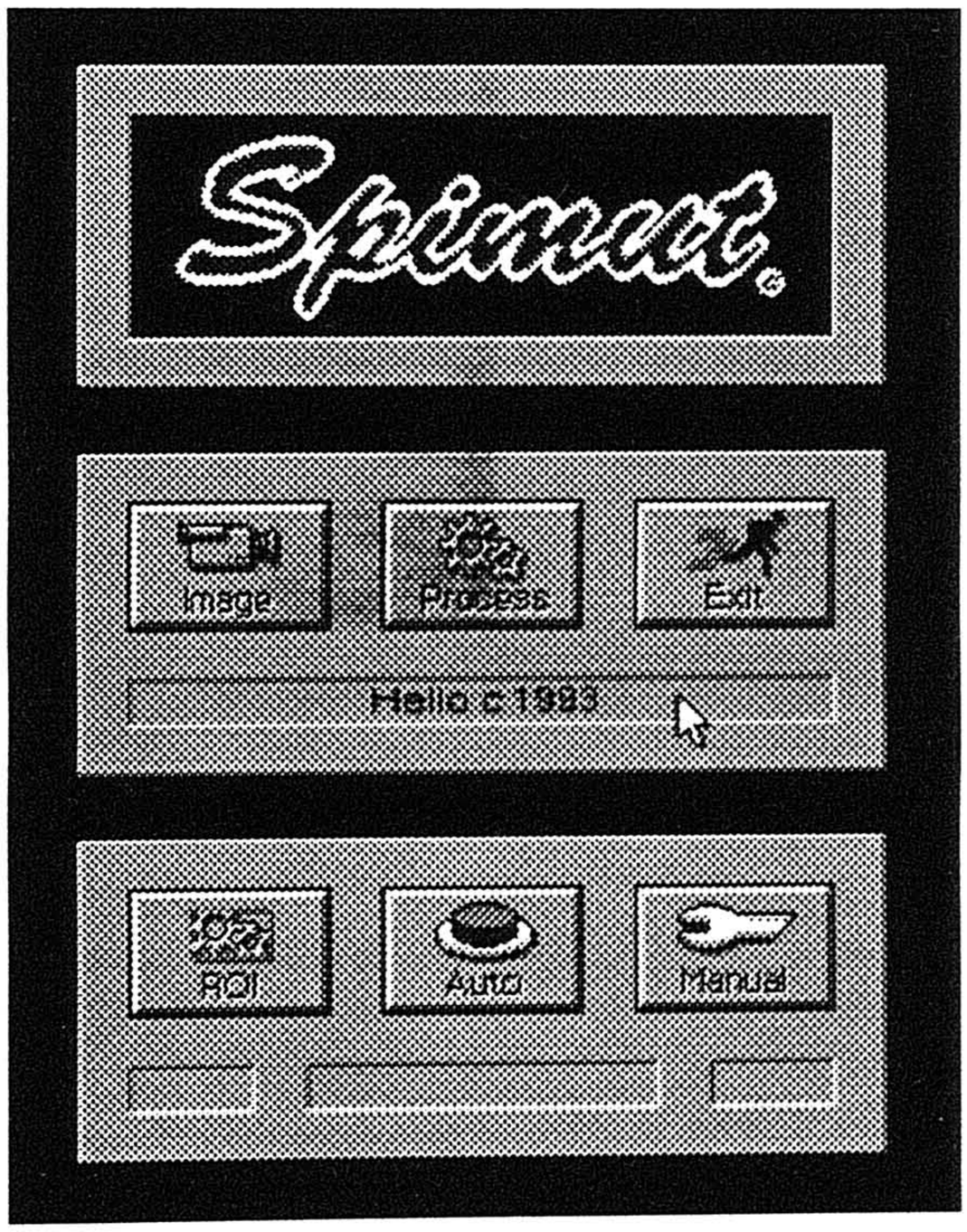


Fig. 39 - Programa SPIMUT

No apêndice A é possível consultar o manual do programa SPIMUT que dá uma visão mais detalhada do seu funcionamento. Nesse manual são também fornecidas indicações sobre a instalação do programa assim como os requisitos mínimos para o seu correcto funcionamento.

O desenvolvimento do programa SPIMUT e de muitos outros programas auxiliares constituiu uma importante parcela em todo o tempo dedicado a este trabalho. Isto porque não só obrigou à elaboração das rotinas de processamento e análise descritas no capítulo anterior, como também de todo um conjunto de rotinas mais simples mas igualmente importantes tais como: criação e cópia de imagens, filtragens, interface com o utilizador, etc

O código fonte é formado por vários ficheiros que se estruturam da forma que a seguir se mostra.

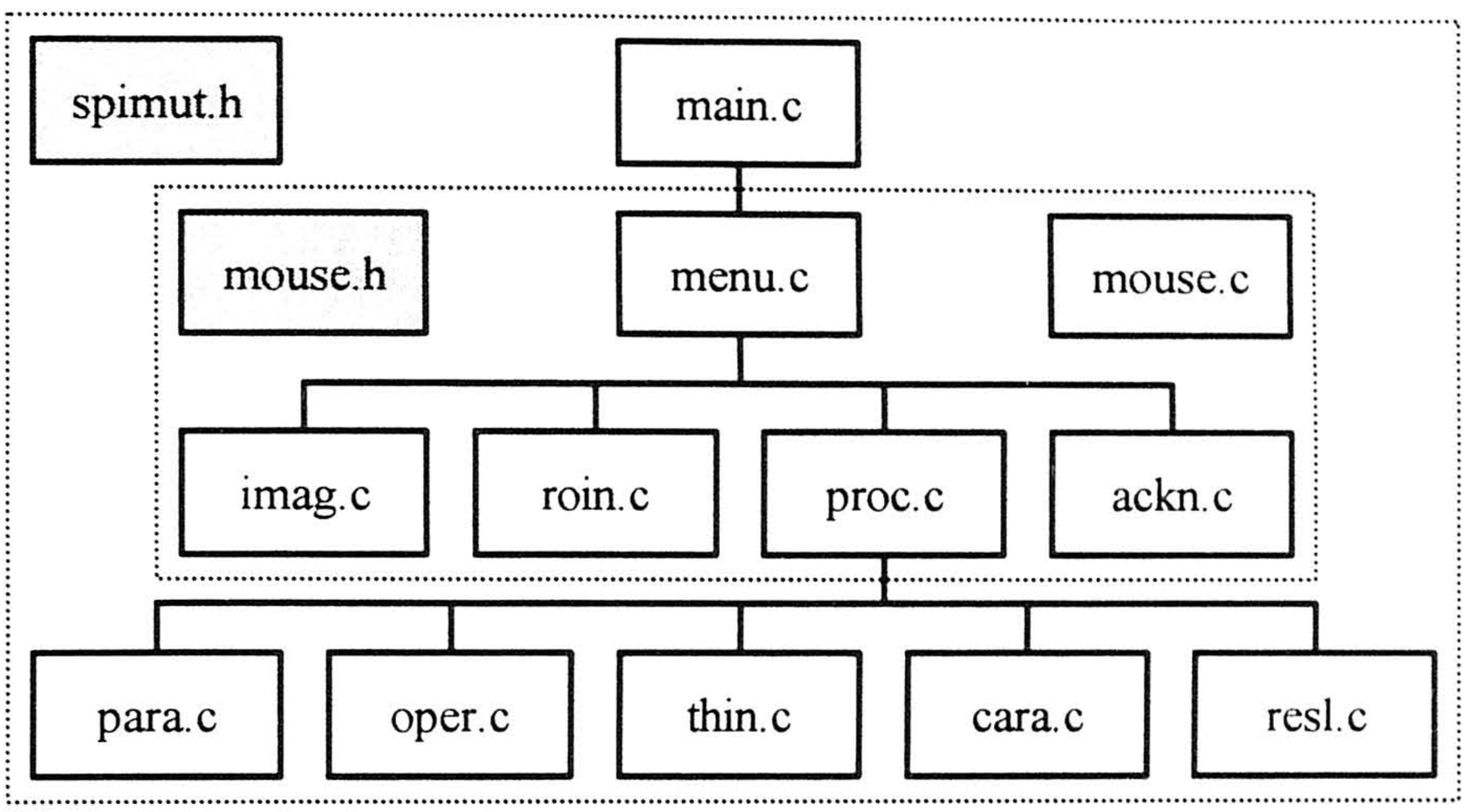
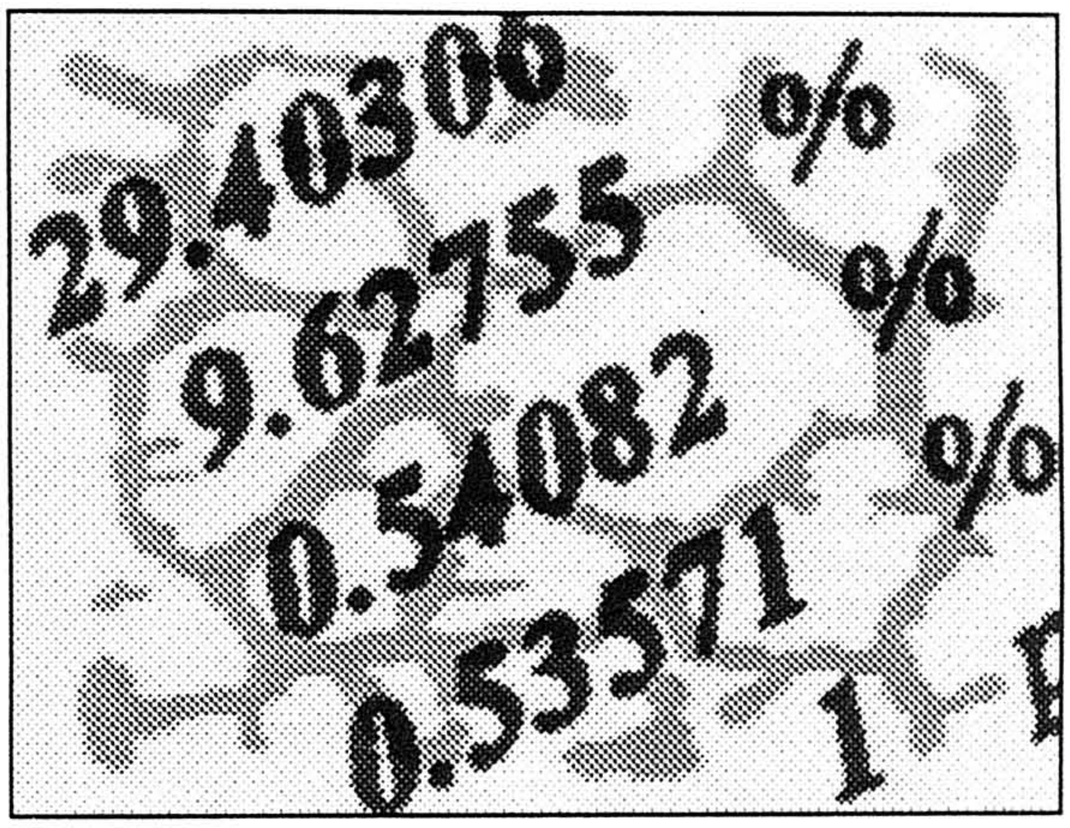


Fig. 40 - Estrutura do programa

Cada um destes ficheiros implementa um conjunto específico de rotinas, tal como seguidamente se descreve.

spimut.h - definições importantes para os restantes ficheiros
main.c - rotinas de inicialização da placa, rato, etc
mouse.h - definições necessárias à utilização do rato
mouse.c - rotinas ligadas ao rato
ackn.c - rotinas de apresentação dos autores
menu.c - criação e manutenção do sistema de menus
imag.c - rotinas ligadas à aquisição, leitura e armazenamento de imagem
roin.c - rotinas de selecção da ROI
proc.c - rotinas associadas ao processamento
para.c - rotinas de remoção da superfície base
oper.c - rotinas várias (criação de imagem, processamento, etc)
thin.c - rotinas de thinning
cara.c - rotinas de extracção de características
resl.c - rotinas de visualização e armazenamento de resultados

No apêndice B é possível consultar as respectivas listagens do código fonte de todos estes ficheiros.



CAPÍTULO 5

RESULTADOS

Neste capítulo serão apresentados alguns resultados da aplicação do algoritmo desenvolvido em imagens simuladas bem como em imagens histeroscópicas reais.

5.1 Imagens simuladas

No sentido de avaliar as performances do algoritmo desenvolvido foram primeiramente utilizadas imagens simuladas o que permitiu ter um maior controlo em determinadas características como o nível de ruído. Para isso foi desenhada uma rede vascular (Fig. 42a), que foi depois sujeita a um conjunto de operações no intuito de se obter uma imagem final com todas as particularidades apresentadas pelas imagens histeroscópicas, designadamente: arestas mal definidas, contraste reduzido, presença de ruído e iluminação não uniforme.

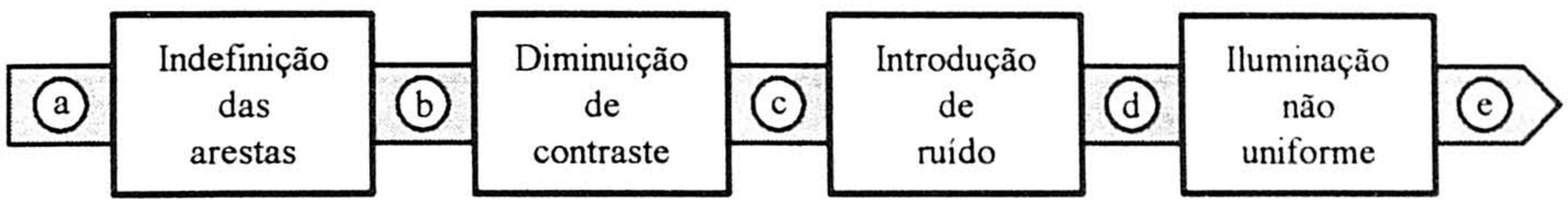


Fig. 41 - Operações necessárias à obtenção das imagens simuladas

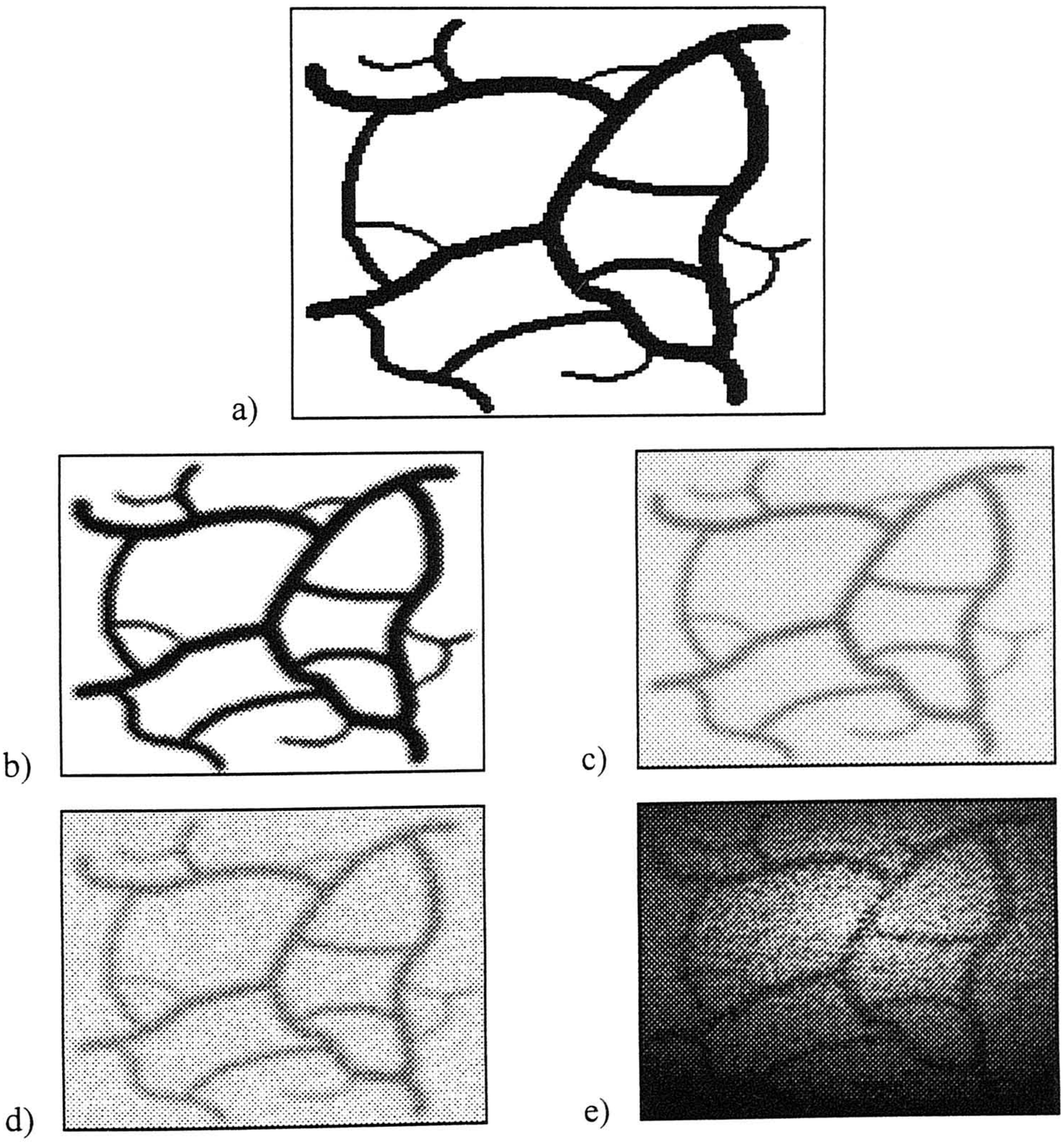


Fig. 42 - a) Rede vascular, b) Arestas mal definidas, c) Contraste reduzido, d) Ruído, e) Iluminação não uniforme

Seguidamente serão apresentadas as redes vasculares detectadas pelo sistema quando a relação sinal ruído é igual a 0 dB e -20 dB.

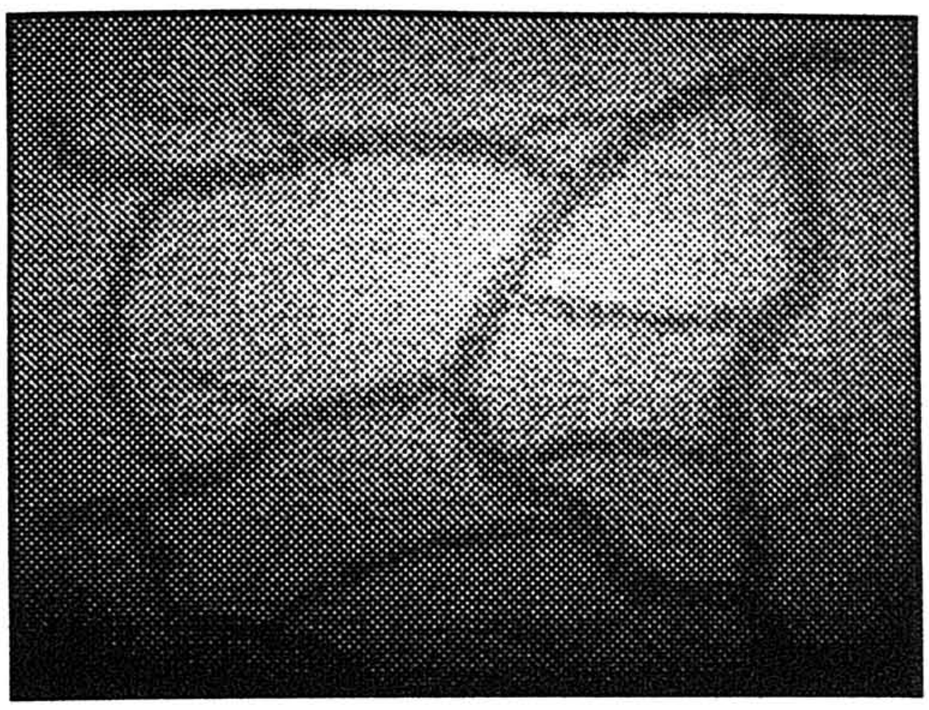
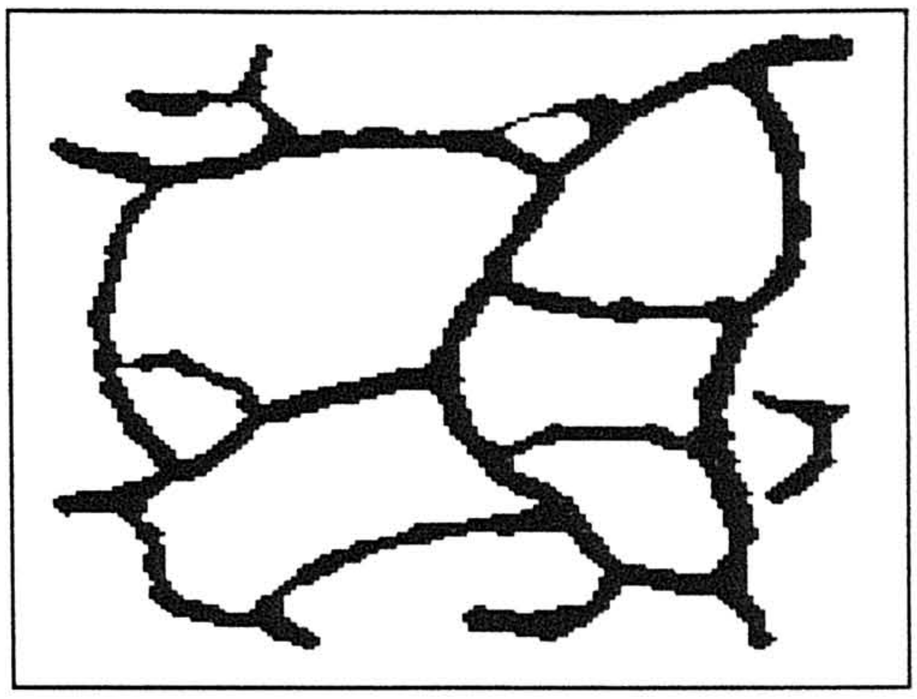
a)  b) 

Fig. 43 - S/N = 0 dB: a) Imagem simulada, b) Rede vascular detectada

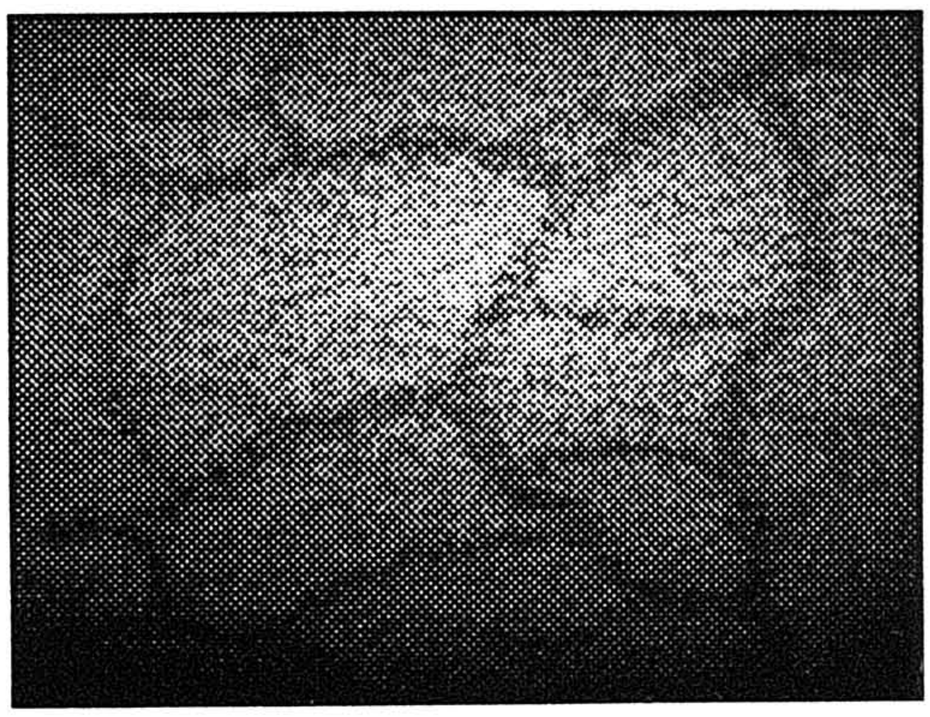
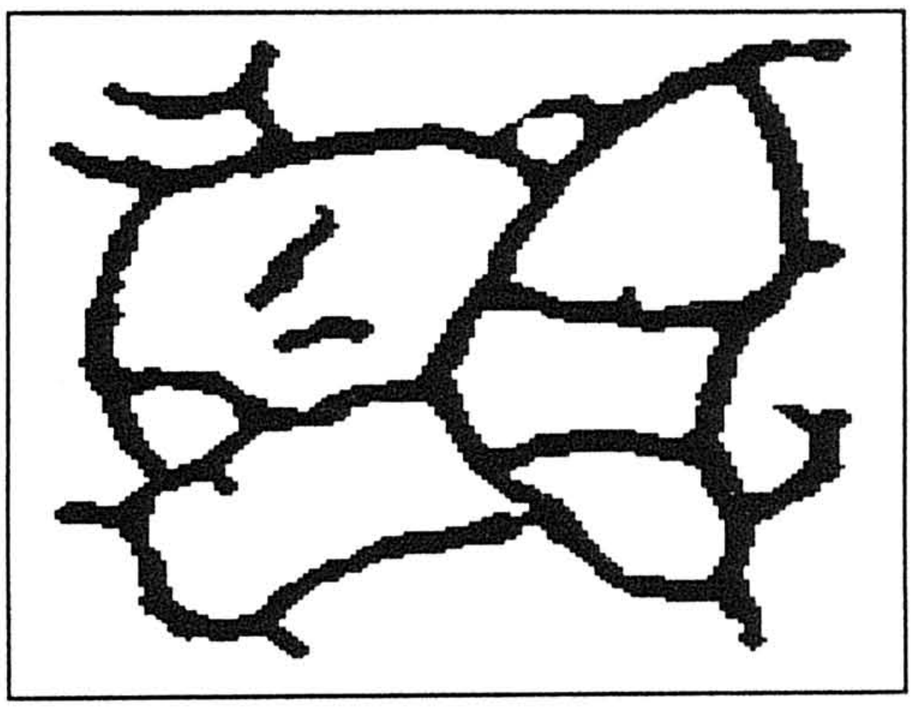
a)  b) 

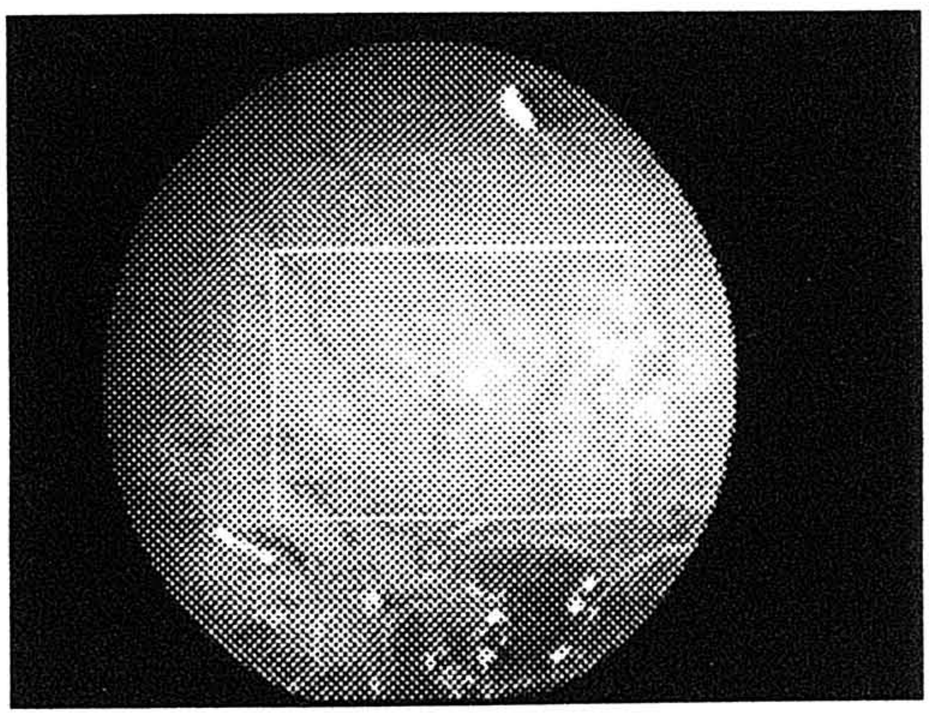
Fig. 44 - S/N = -20 dB: a) Imagem simulada, b) Rede vascular detectada

De notar que mesmo na situação mais desfavorável (S/N = -20 dB) onde o nível de ruído é duas vezes maior do que o nível do sinal, a rede vascular detectada é bastante próxima da rede original.

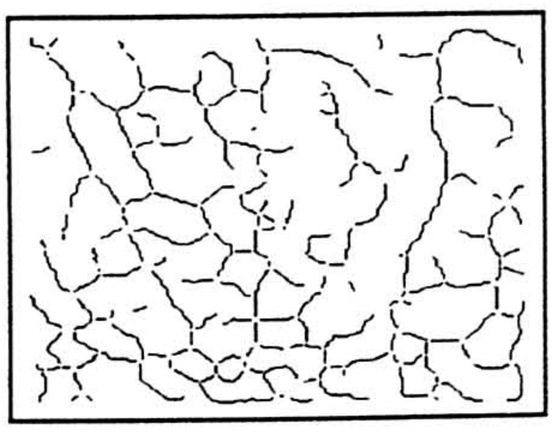
5.2 Imagens reais

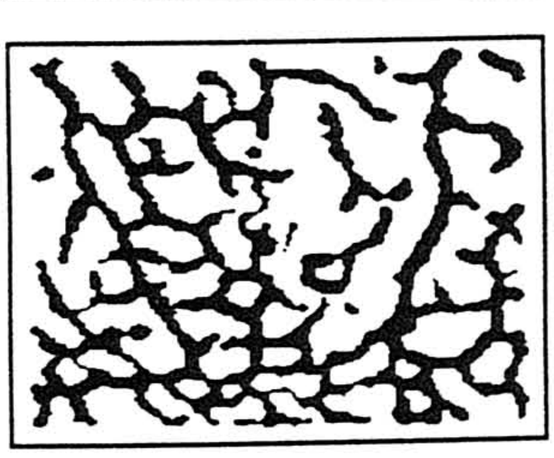
As performances do algoritmo não ficariam, como é óbvio, completamente avaliadas se este não fosse aplicado a imagens histeroscópicas reais para o qual foi desenvolvido. Seguidamente serão apresentados os resultados do processamento e análise efectuados em duas imagens histeroscópicas representativas de duas situações distintas em termos de desenvolvimento do endométrio: normal e deficiente (indiciando neste caso uma situação de infertilidade).

Embora seja imperiosa a análise estatística dos resultados obtidos a partir de uma gama de imagens histeroscópicas mais extensa, é desde já possível constatar que determinadas características apresentam valores bastante díspares em cada uma das duas situações aqui analisadas. Com efeito a área e perímetro vascular é muito menor na situação de infertilidade, assim como o número relativo de segmentos e nodos. De notar a maior uniformidade de distribuição das características pela imagem quando o desenvolvimento do endométrio é normal como atestam os elevados valores de homogeneidade.

a) 

b) 

c) 

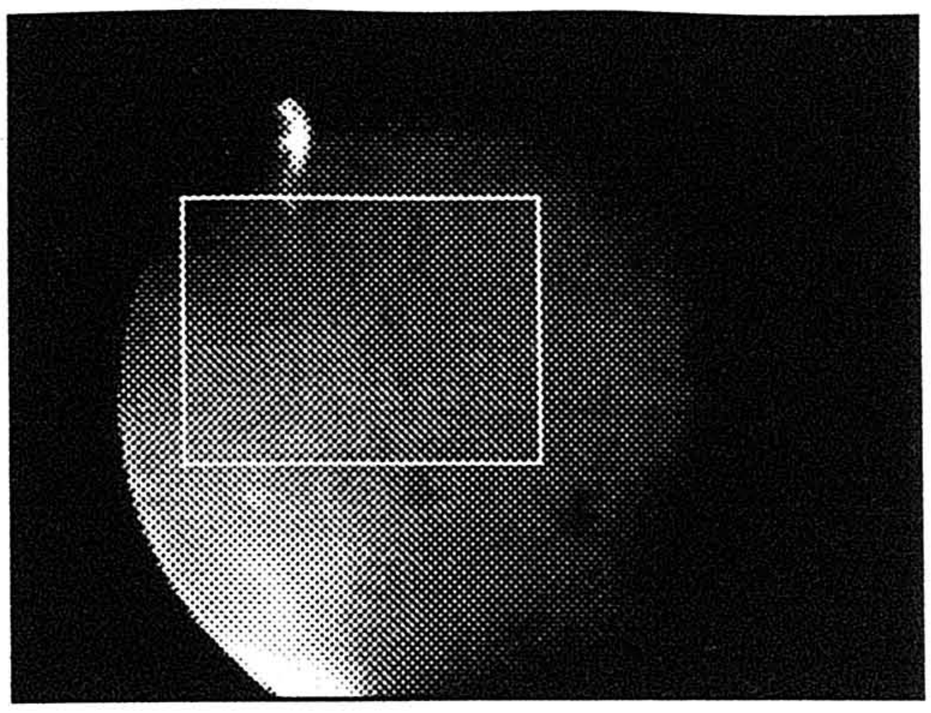
d) 

e)

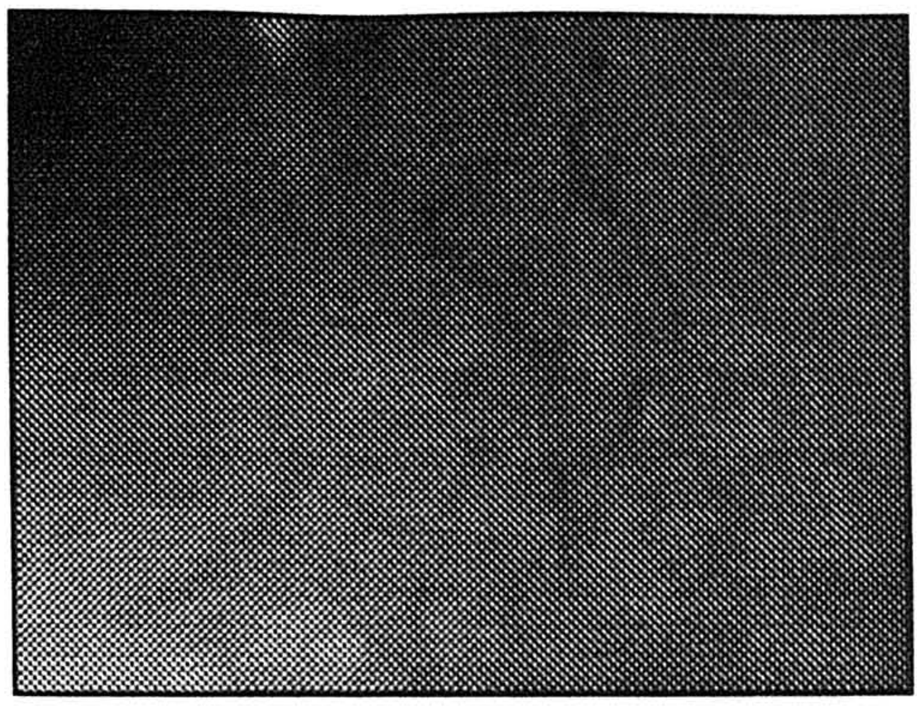
Área vascular	***30.93906***	*%*
Perímetro vascular	***11.05817***	*%*
Número de segmentos	***0.55402***	*%*
Número de nodos	***0.54017***	*%*
Calibre mínimo	***1***	*p*
Calibre médio	***4***	*p*
Calibre máximo	***7***	*p*
Comprimento mínimo	***2***	*p*
Comprimento médio	***10***	*p*
Comprimento máximo	***37***	*p*
Homogeneidade de área	***2.73538***	
Homogeneidade de perímetro	***2.22868***	
Homogeneidade de calibre	***3.56807***	
Homogeneidade de perímetro	***1.15240***	

Fig. 45 - Desenvolvimento normal: a) Imagem histeroscópica, b) ROI, c) Segmentos classificados como vasos, d) Rede reconstruída, e) Características extraídas

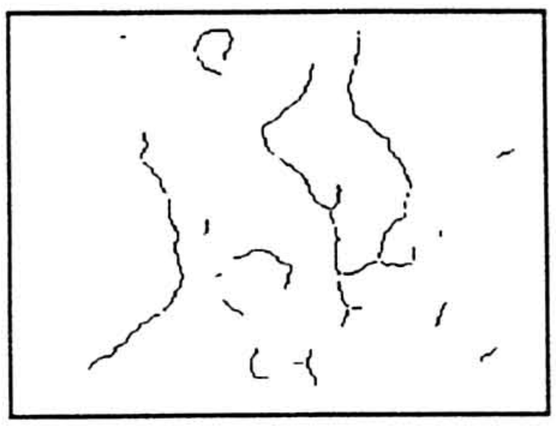
a)



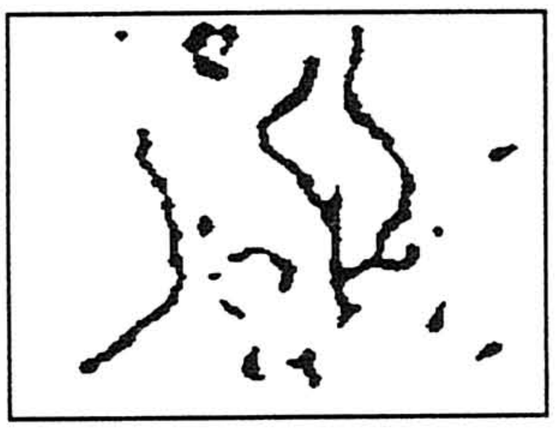
b)



c)



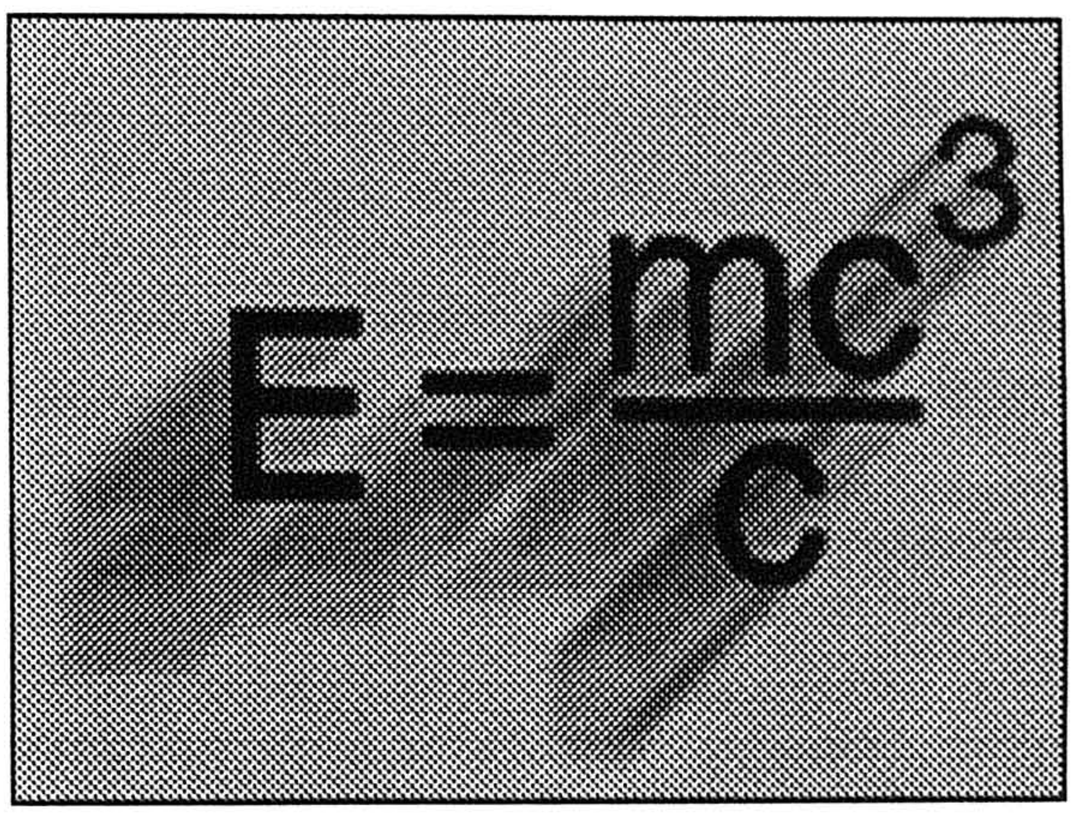
d)



e)

Área vascular	***8.12742***	*%*
Perímetro vascular	***3.48476***	*%*
Número de segmentos	***0.09972***	*%*
Número de nodos	***0.12742***	*%*
Calibre mínimo	***3***	*p*
Calibre médio	***5***	*p*
Calibre máximo	***7***	*p*
Comprimento mínimo	***2***	*p*
Comprimento médio	***16***	*p*
Comprimento máximo	***62***	*p*
Homogeneidade de área	***1.76685***	
Homogeneidade de perímetro	***0.98711***	
Homogeneidade de calibre	***1.05338***	
Homogeneidade de perímetro	***1.02112***	

Fig. 46 - Desenvolvimento deficiente: a) Imagem histeroscópica, b) ROI, c) Segmentos classificados como vasos, d) Rede reconstruída, e) Características extraídas



CAPÍTULO 6

CONCLUSÕES

Neste capítulo são indicadas algumas considerações finais sobre o trabalho desenvolvido, sendo apresentados aspectos passíveis de serem melhorados em futuras evoluções do sistema.

Como se viu no capítulo anterior os resultados obtidos pelo algoritmo em geral e pela operação de detecção de redes vasculares em particular foram bastante satisfatórios não obstante as características algo adversas apresentadas pelas imagens. Este bom comportamento manteve-se para a grande maioria das restantes imagens histeroscópicas.

Para além das boas performances este algoritmo de detecção de redes vasculares apresenta um conjunto de outras características que o distinguem face a algoritmos afins, e que são as seguintes:

- ***rapidez*** - num computador 486DX a 33 MHz uma ROI de 256 por 256 pixels é inteiramente processada e as suas características extraídas em cerca de 15 segundos.
- ***abordagem simultaneamente local e global*** - enquanto que operações como a detecção de mínimos locais são basicamente locais (ao nível do pixel), operações como a classificação são de âmbito global (ao nível do segmento).
- ***exigência em termos de qualidade de imagem*** - trata-se de um algoritmo pouco exigente em termos de definição das arestas dos vasos assim como da continuidade dos mesmos.
- ***flexibilidade*** - a alteração do valor dos diversos parâmetros que integram o algoritmo (*r*, *R*, *mpc*, *msc*) permite ajustá-lo a outro tipo de imagens (com diferentes características).

O sistema encontra-se instalado no Serviço de Ginecologia do Hospital de S. João do Porto tendo merecido pareceres extremamente favoráveis por parte do Prof. J. P. Marques de Sá bem como do Dr. J. L. Silva Carvalho. É contudo possível indicar um conjunto de aspectos que poderão ser alvo de futuras evoluções no sentido de melhorar ainda mais as actuais performances.

A operação de detecção de mínimos locais poderá evoluir no sentido de a medição do contraste de um pixel ser função de todos os pixels pertencentes a uma vizinhança em torno dele em vez de se restringir às actuais duas coroas.

A operação de decomposição em segmentos poderá ser alterada por forma a que o resultado se aproxime mais do apresentado na Fig. 47c do que o actual (Fig. 47b). Com isto caminharemos no sentido de termos uma correspondência directa entre um segmento e um vaso.

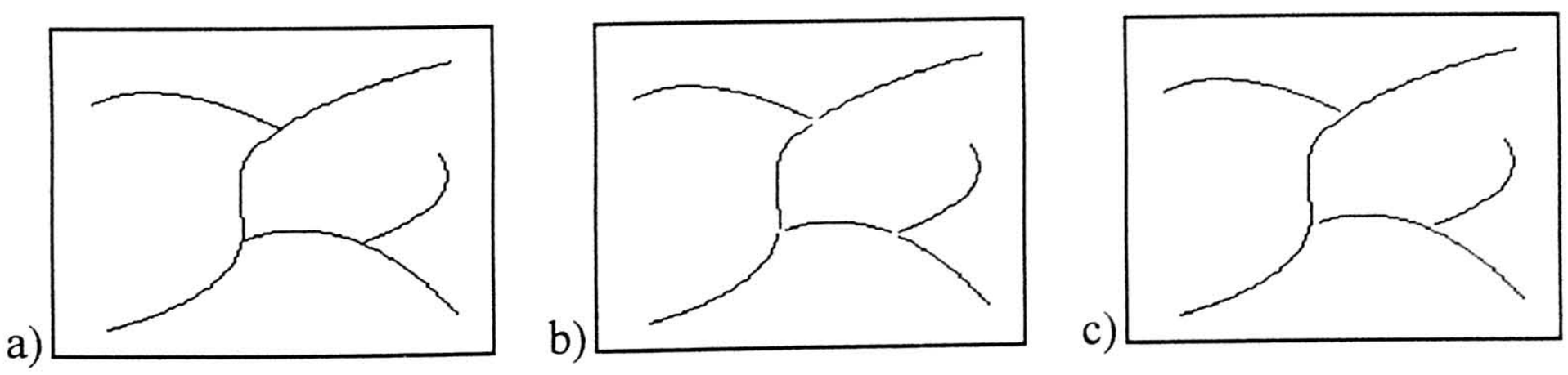


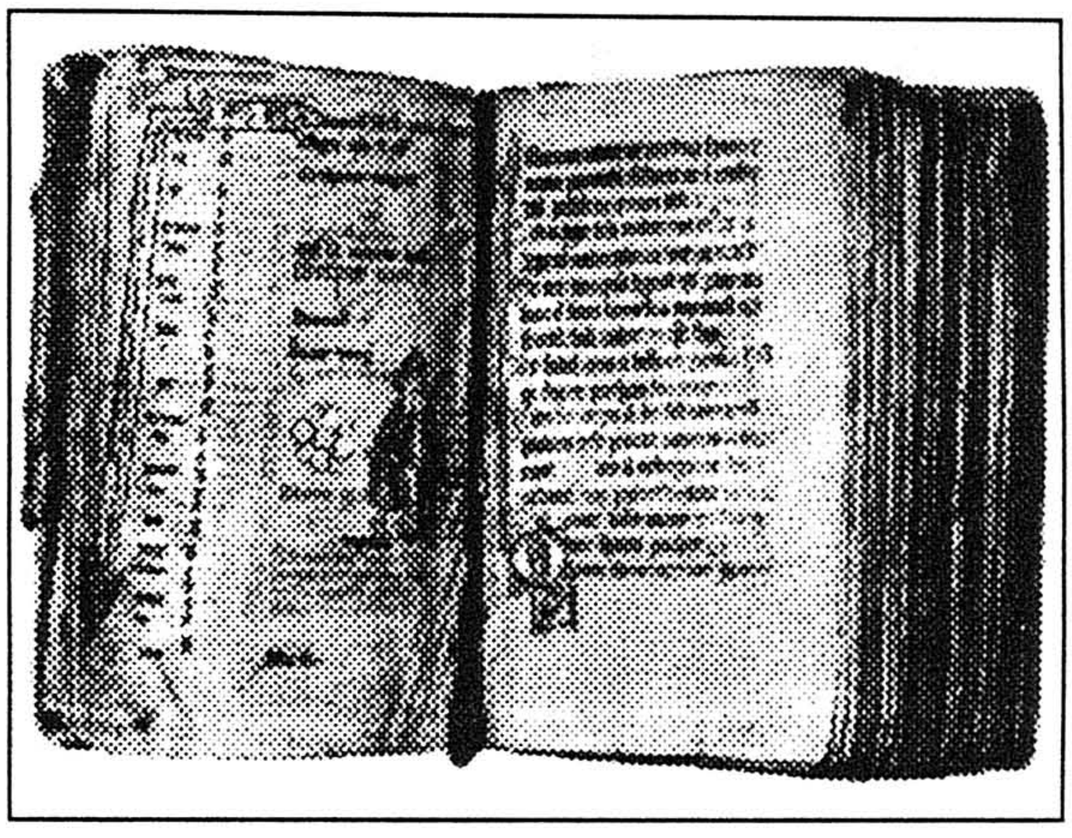
Fig. 47 - a) Esqueleto, b) Resultado actual, c) Resultado futuro

A classificação dos segmentos efectuada presentemente por um classificador muito simples poderá ser melhorada utilizando um classificador mais versátil como aquele que seguidamente se apresenta.

$$k_1.Intensity + k_2.Contrast + k_3.Intensity.Contrast < k$$

Por último a detecção das arestas quando da reconstrução da rede a partir dos segmentos classificados como vasos, poderá ser tornada mais sofisticada pela utilização conjugada de métodos de detecção de arestas como os de Canny ou Deriche.

Espero que o sistema computacional desenvolvido venha a constituir uma importante ferramenta de auxílio ao diagnóstico de situações de infertilidade e cancro uterino, conferindo assim uma maior fiabilidade à histeroscopia como técnica de diagnóstico. Espero ainda que o algoritmo de processamento descrito neste trabalho possa tornar-se num método alternativo de detecção de redes vasculares especialmente se as imagens apresentarem características adversas à utilização dos métodos usuais. Tenho contudo a certeza que todo o trabalho desenvolvido ao longo dos últimos dois anos me forneceu um vasto conjunto de conhecimentos ao nível do processamento e análise de imagem.



CAPÍTULO 7

BIBLIOGRAFIA

Neste capítulo será listada a bibliografia mais significativa de entre aquela que foi utilizada nas fase de pesquisa, elaboração e teste deste trabalho.

[1] Gynaecological Endoscopy, Alan G. Gordon, B. Victor Lewis, Chapman and Hall Medical, 1988
[2] Hysteroscopes, STORZ - The World of Endoscopy, 4th edition, 1/90
[3] Basic Hysteroscopy Course Book, HTC Hysteroscopy Training Center, November 1990
[4] Gray Level Thresholding in Badly Illuminated Images, J. R. Parker, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13, No. 8, August 1991, pp 813-819.
[5] Using Canny´s Criteria to Derive a Recursively Implemented Optimal Edge Detector, R. Deriche, International Journal on Computer Vision, Vol. 1, No 2, May 1987, pp 167-187.
[6] A Computer Approach to Edge Detection, J. Canny, IEEE Transactions PAMI, 8, 1986, pp 679-698
[7] Detection of Blood Vessels in Retinal Images Using Two-Dimensional Matched Filters, S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, M. Goldbaum, IEEE Transactions on Medical Imaging, Vol. 8, No 3, September 1989, pp 263-269.
[8] An Artificial Vision System for X-Ray Images of Human Coronary Trees, G. Coppini, M. Demi, G. Valli, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, No. 2, February 1993, pp 156-162.
[9] Automated Identification of Vessels Contours in Coronary Arteriograms by an Adaptative Tracking Algorithm, Ying Sun, IEEE Transactions on Medical Imaging, Vol. 8, No 1, March 1989, pp 78-88.
[10] Automated Tracking of the Vascular Tree in DSA Images Using a Double-Square-Box Region-of-Search Algorithm, K. R. Hoffmann, Kunio Doi, Heang-Ping Chan, L. Fencil, H. Fujita, A. Muraki, SPIE Vol. 626, Medicine XIV/PACS IV, 1986, pp 326-333
[11] Digital Picture Processing, A. Rosenfeld, A. C. Kak, Academic Press, 1976
[12] Digital Picture Processing, A. Rosenfeld, A. C. Kak, 2nd ed., Academic Press, 1982
[13] Digital Image Processing, Rafael C. Gonzalez, Paul Wintz, Addison-Wesley Publishing CompanyAcademic Press, 2nd edition, 1987
[14] Baseline Fluctuations Removal from Impedance Cardiographic Signals, H. M. Mendonça, J. Jossinet, J. P. Marques de Sá, Innovation et Technologie en Biologie et Medicine, Vol. 13, No. 2, 1992, pp 183-187.
[15] User Manual for DT-2851, High Resolution Frame Grabber, Document UM-05234-E, Data Translation Inc
[16] User Manual for DT-IRIS, Subroutine Library, Document UM-05633-D, Data Translation Inc



APÊNDICE A

MANUAL DO PROGRAMA

Neste apêndice é apresentado o manual do programa SPIMUT.
Neste manual são abordadas questões relativas à instalação do programa, requisitos mínimos e modo de utilização.

ÍNDICE

1 Descrição 2

2 Requisitos 3
2.1 Hardware 3
2.2 Software 3

3 Instalação 4

4 Funcionamento 5
4.1 Arranque 5
4.2 Erros 6
4.2.1 Erros durante o arranque 6
4.2.2 Erros durante o funcionamento 6
4.3 Princípio de funcionamento 7
4.4 Menu principal 8
4.4.1 Botão IMAGE 8
4.4.2 Botão PROCESS 8
4.4.3 Botão EXIT 8
4.5 Menu de imagem 9
4.5.1 Botão LOAD 9
4.5.2 Botão ACQUIRE 10
4.5.3 Botão SAVE 10
4.6 Menu de processamento 11
4.6.1 Botão ROI 11
4.6.2 Botão AUTO 12
4.6.3 Botão MANUAL 14
4.6.4 Características 14
4.7 Menu de resultados 16
4.7.1 Botão NAME 16
4.7.2 Botão DISK 16
4.7.3 Botão PRINTER 17

1 DESCRIÇÃO

O programa SPIMUT (Sistema de Processamento de Imagens Uterinas) destina-se a efectuar o processamento e análise de imagens histeroscópicas uterinas previamente armazenadas em vídeo.

Trata-se de um programa capaz de obter de uma forma precisa, objectiva e quantitativa um conjunto de características de interesse clínico sobre a rede vascular uterina. Essas características podem ser depois guardadas em disco para posterior processamento estatístico.

Pretende-se que o programa desenvolvido seja uma ferramenta simples do ponto de vista de utilização mas simultaneamente poderosa no auxílio ao diagnóstico de determinadas situações patológicas como a infertilidade e o cancro do útero.

2 REQUISITOS

Para que o programa SPIMUT funcione adequadamente deverão ser satisfeitos determinados requisitos quer ao nível do equipamento (hardware) quer ao nível dos programas (software).

2.1 Hardware

O equipamento necessário ao correcto funcionamento do sistema desenvolvido é o seguinte:

- computador compatível PC/AT
- placa de aquisição de imagem DT2851 da Data Translation
- videogravador
- monitor auxiliar

Por sua vez o computador acima mencionado deverá cumprir as seguinte especificações:

- pelo menos 512 Kbytes de RAM
- placa de vídeo VGA
- rato compatível Microsoft

Embora não seja imperativa a utilização de um computador com processador 486 ou coprocessador aritmético, esta recomenda-se fortemente. Isto porque ao longo do processamento são efectuadas operações bastante pesadas em termos de cálculo pelo que a sua não utilização conduzirá a tempos de processamento demasiado elevados.

2.2 Software

No que diz respeito ao software, é requerido que no computador esteja devidamente instalado:

- sistema operativo MS-DOS versão 3.3 ou superior
- *device driver* da placa DT2851
- *device driver* do rato
- programa SPIMUT

3 INSTALAÇÃO

Antes de se efectuar a instalação do programa SPIMUT recomenda-se que seja criada uma cópia da disquete de instalação. Este processo é efectuado da seguinte maneira:

- introduzir a disquete original de instalação no *drive a*
- digitar `diskcopy a: a:`↵
- seguir as instruções que vão surgindo no monitor

A disquete original de instalação deverá ser guardada em local seguro por forma a ser utilizada caso a sua cópia se danifique. Nesse caso deveremos proceder à criação de nova cópia a partir da disquete original.

A instalação do programa SPIMUT faz-se simplesmente correndo o programa *install* existente na cópia da disquete de instalação. Para tal basta proceder da seguinte forma:

- introduzir a disquete no *drive a*
- digitar `a:install`↵

O programa de instalação irá criar no disco duro do computador um directório *spimut* onde são colocados todos os ficheiros necessários à execução do programa SPIMUT, e que são:

- README.1ST ficheiro contendo informações não presentes neste manual
- SPIMUT.EXE programa principal
- PICS.DAT ficheiro que contém dados sobre o logotipo, botões, etc
- HELVB.FON ficheiro com informações relativas à fonte de caracteres

4 FUNCIONAMENTO

4.1 Arranque

Uma vez realizada a instalação do programa SPIMUT é possível arrancar com o seu funcionamento. Este é efectuado fazendo o seguinte:

- digitar `cd \spimut↵`
- digitar `spimut↵`

Deverá então surgir no monitor do computador uma imagem igual à que se encontra na Fig. 1. A partir deste momento o programa SPIMUT está pronto a ser utilizado.

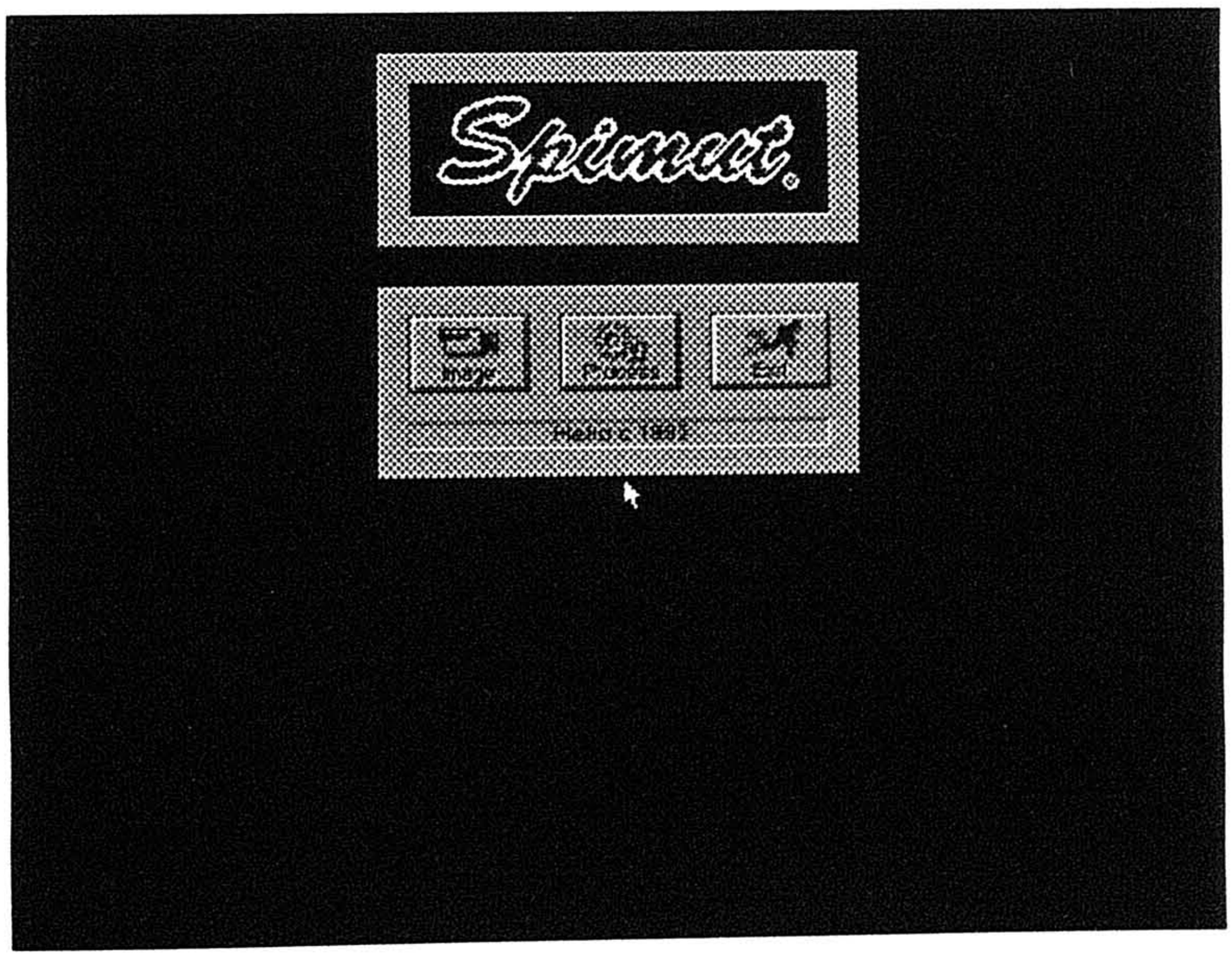


Fig. 1 - Programa SPIMUT

4.2 Erros

Quer durante o arranque quer durante o funcionamento do programa podem ocorrer erros. Estes erros têm causas e consequências distintas pelo que serão analisados separadamente nas seguintes secções.

4.2.1 Erros durante o arranque

Este tipo de erros, como o nome indica, ocorre durante o arranque do programa impedindo assim o seu funcionamento. É então apresentada uma mensagem de erro cujas causas são as seguintes.

- **Error: DT2851 not present**
  placa de aquisição ou *device driver* respectivo incorrectamente instalados
- **Error: Mouse not present**
  rato ou *device driver* respectivo incorrectamente instalados
- **Error: VGA not present**
  modo de vídeo VGA não suportado pelo computador
- **Error: Can't register fonts**
  ficheiro HELVB.FON não disponível
- **Error: Can't register pics**
  ficheiro PICS.DAT não disponível
- **Error: Allocation memory**
  memória disponível insuficiente

4.2.2 Erros durante o funcionamento

Os erros que ocorrem durante o funcionamento não terminam o programa, apenas sinalizam a impossibilidade de execução de determinadas operações. Tal como a Fig. 2 mostra um erro deste tipo é indicado pela presença da mensagem ERROR na parte inferior do menu principal acompanhado por um som de alerta.

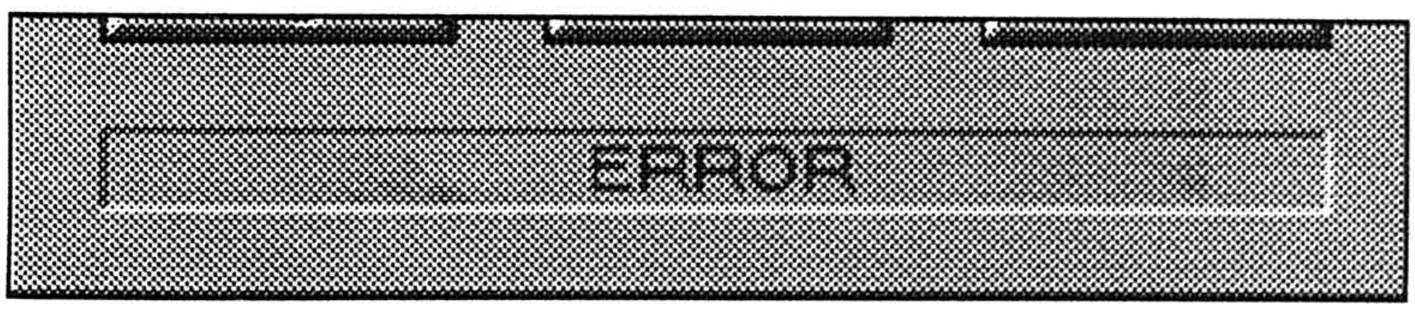


Fig. 2 - Mensagem de erro de funcionamento

De entre as situações que podem dar origem a este tipo de erros salientam-se as seguintes:

- leitura ou escrita de imagens de forma incorrecta
- armazenamento dos resultados de forma incorrecta
- escolha de nomes inadequados para os ficheiros
- erro no envio de resultados para a impressora
- falta de memória

4.3 Princípio de funcionamento

O princípio de funcionamento do programa SPIMUT é a activação de botões. Em qualquer momento o utilizador apenas terá que premir de entre os vários botões visíveis no monitor aquele que, pelas suas características, é o mais adequado à realização da operação pretendida.

A activação de um botão faz-se por intermédio do cursor do rato sempre que este se encontra disponível (Fig. 3). Para tal basta colocar o cursor sobre o botão que se deseja activar e carregar no botão esquerdo do rato. Como consequência desta acção o botão no monitor desloca-se, iniciando-se de seguida a operação correspondente.

Como se pode constatar na Fig. 4 os botões possuem as seguintes duas características:

- um nome associado à operação a que dão origem (PROCESS)
- um desenho que simboliza essa mesma operação (um par de rodas dentadas)

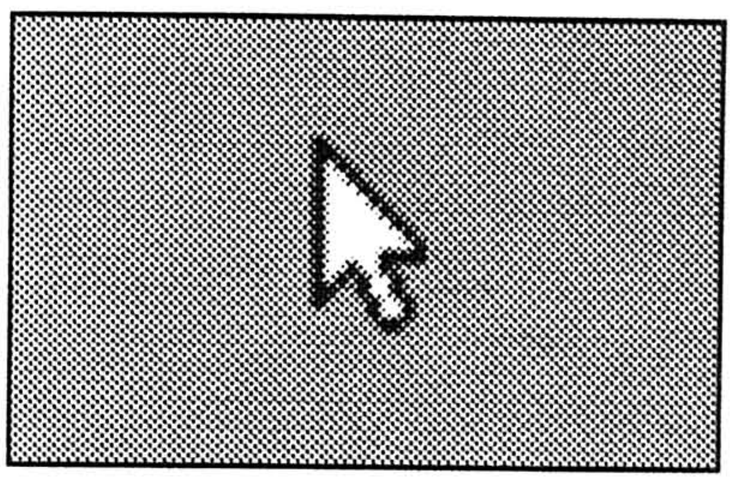


Fig. 3 - Cursor do rato

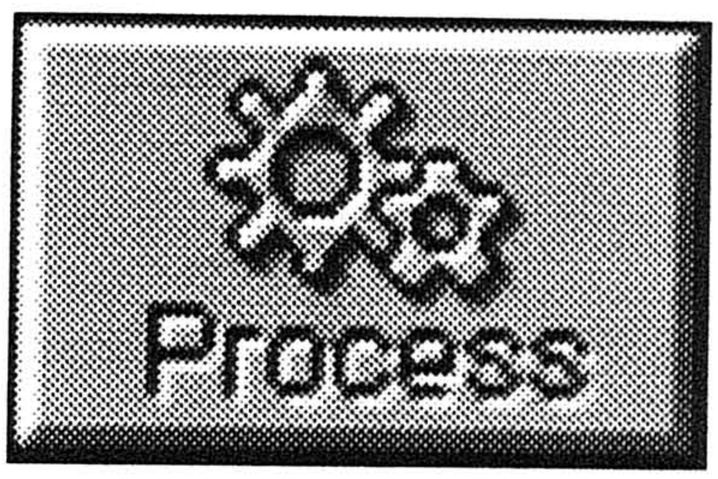


Fig. 4 - Exemplo de um botão

O programa SPIMUT encontra-se estruturado em menus. Cada um destes menus é composto por três botões associados a igual número de operações. O menu principal encontra-se sempre presente, sendo possível a partir dele chegar aos restantes.

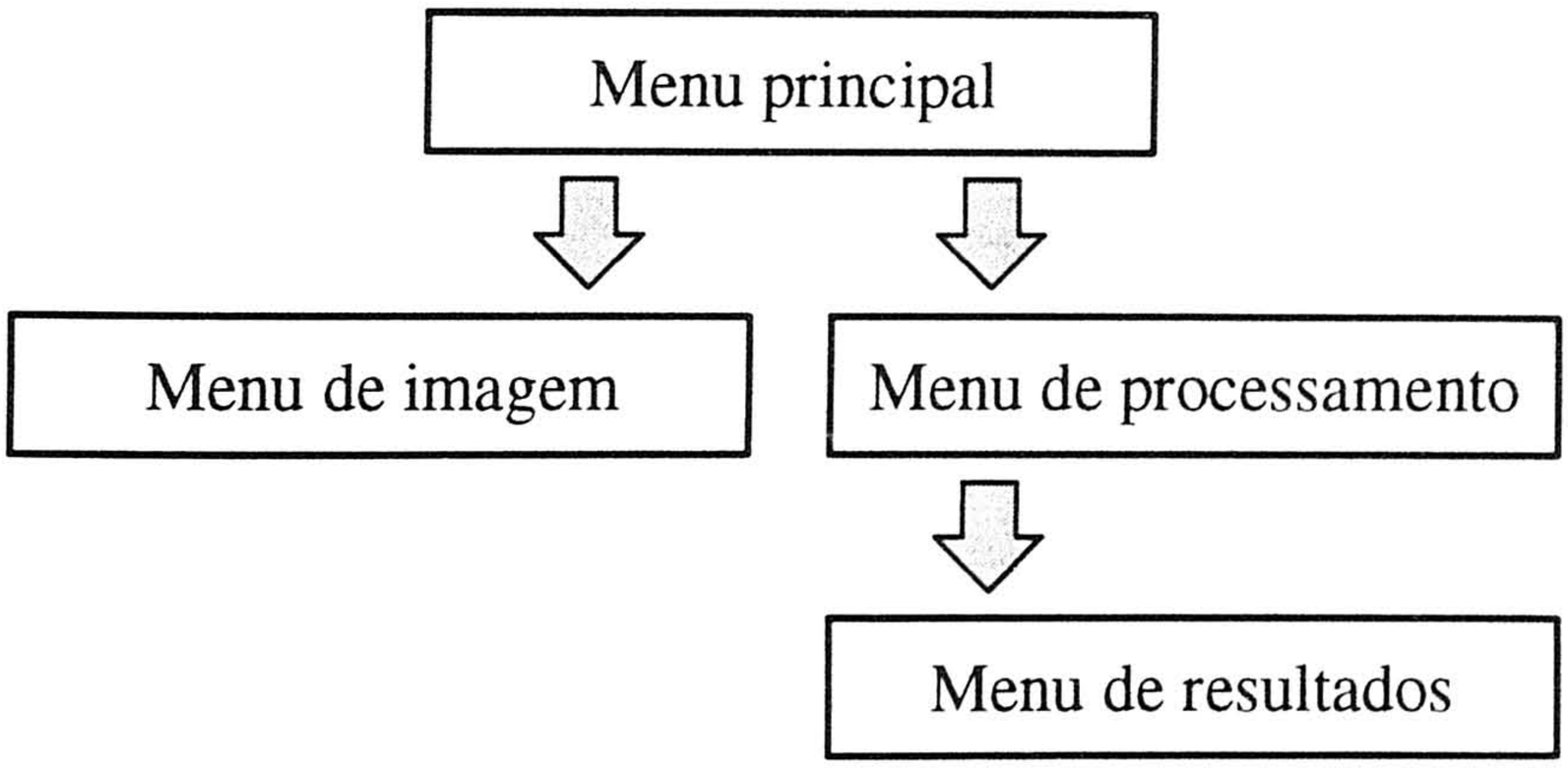


Fig. 5 - Estrutura de menus

Nas próximas secções serão apresentados cada um destes menus, bem como os botões que os compõem.

4.4 Menu principal

Como foi já referido o menu principal dá acesso aos restantes menus. Como se vê na Fig. 6, trata-se de um menu composto pelos seguintes três botões:

- IMAGE
- PROCESS
- EXIT

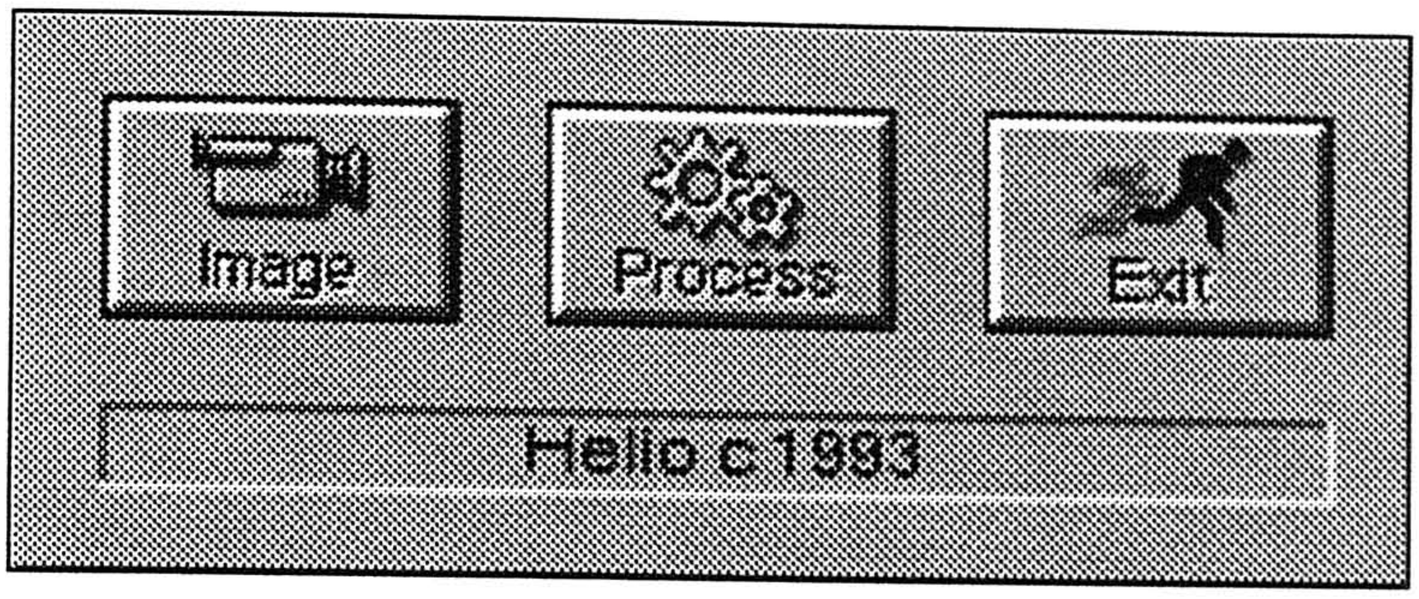


Fig. 6 - Menu principal

4.4.1 Botão IMAGE

A activação deste botão faz surgir o menu de imagem que será descrito mais tarde. Ficam então acessíveis os botões de leitura, aquisição e escrita de imagens.

4.4.2 Botão PROCESS

Por seu lado, a activação do botão PROCESS faz surgir o menu de processamento onde decorrem as operações ligadas ao processamento das imagens.

4.4.3 Botão EXIT

Finalmente o botão EXIT termina a execução do programa retornando ao sistema operativo.

4.5 Menu de imagem

O menu de imagem dá acesso aos três botões que procedem à leitura, aquisição e escrita das imagens a processar. Esses botões são os seguintes:

- LOAD
- ACQUIRE
- SAVE

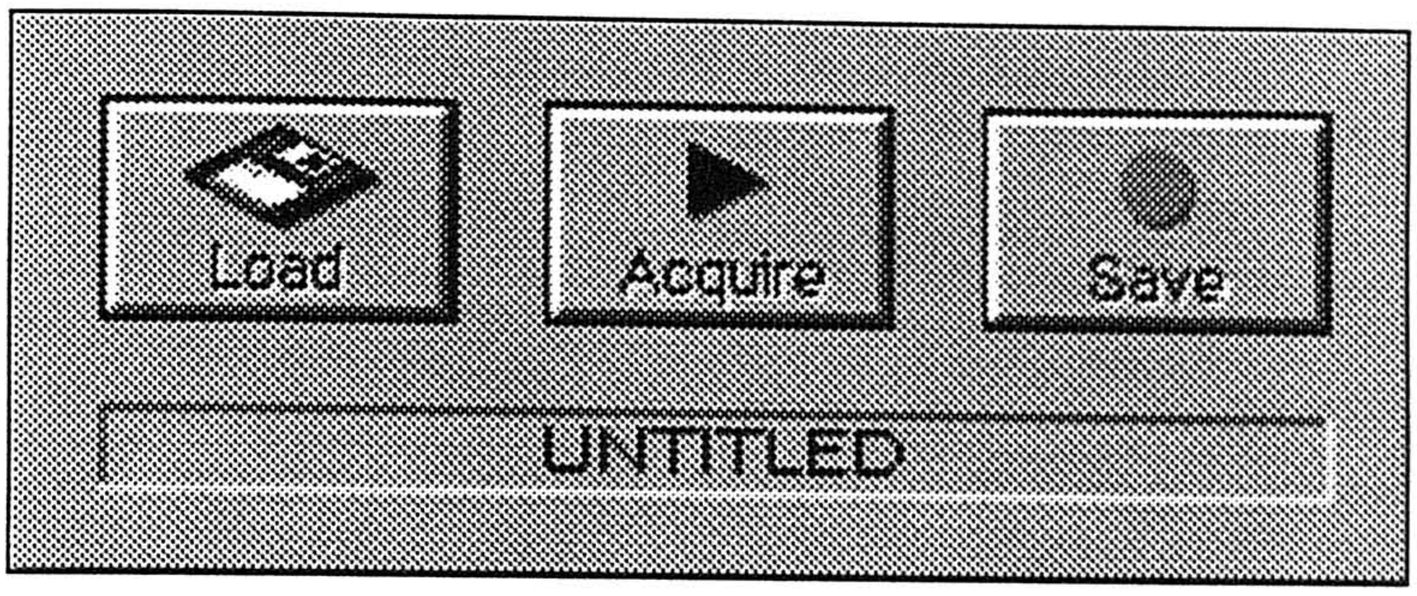


Fig. 7 - Menu de imagem

Quando se acede a este menu, surge no monitor auxiliar a imagem armazenada na memória do computador. Na parte inferior do menu é colocado o nome da imagem. Após o arranque não existindo ainda qualquer imagem armazenada o nome utilizado é *untitled*.

4.5.1 Botão LOAD

O botão LOAD serve para ler uma imagem previamente guardada em disco. Consequentemente o utilizador é questionado acerca do nome e da localização da imagem (*filename*). Como pode ser visto na Fig. 8 o nome da última imagem surge já inserido pelo que o utilizador deverá apenas fazer as alterações que achar adequadas.

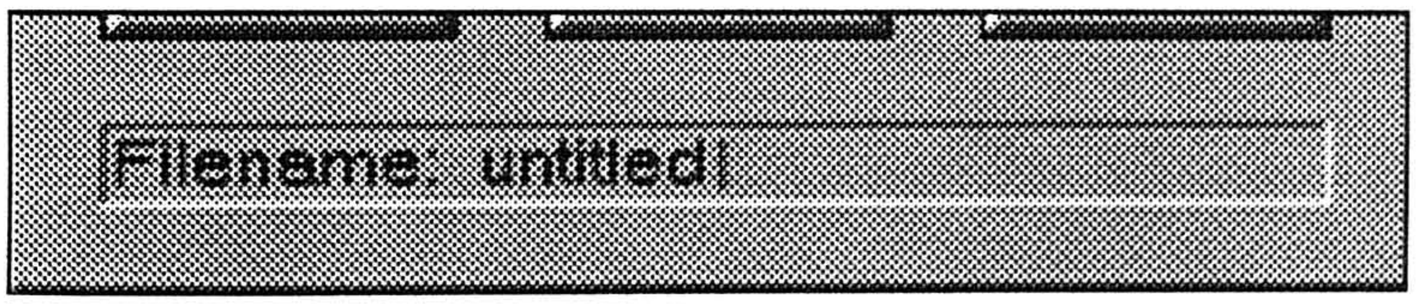


Fig. 8 - Introdução do nome e da localização da imagem

Uma vez efectuada a leitura apenas o nome da imagem aparece na zona inferior do menu.

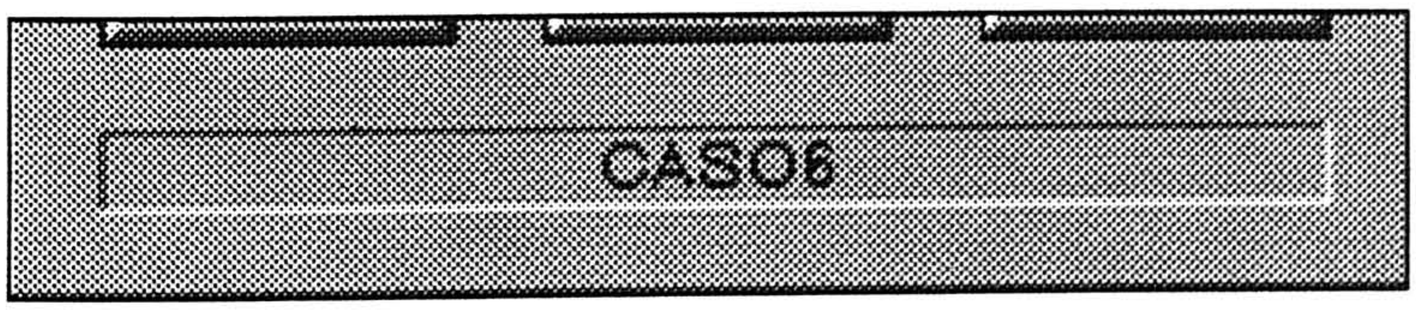


Fig. 9 - Nome da imagem

Por sua vez o monitor auxiliar apresenta a imagem que foi lida por ordem expressa do utilizador.

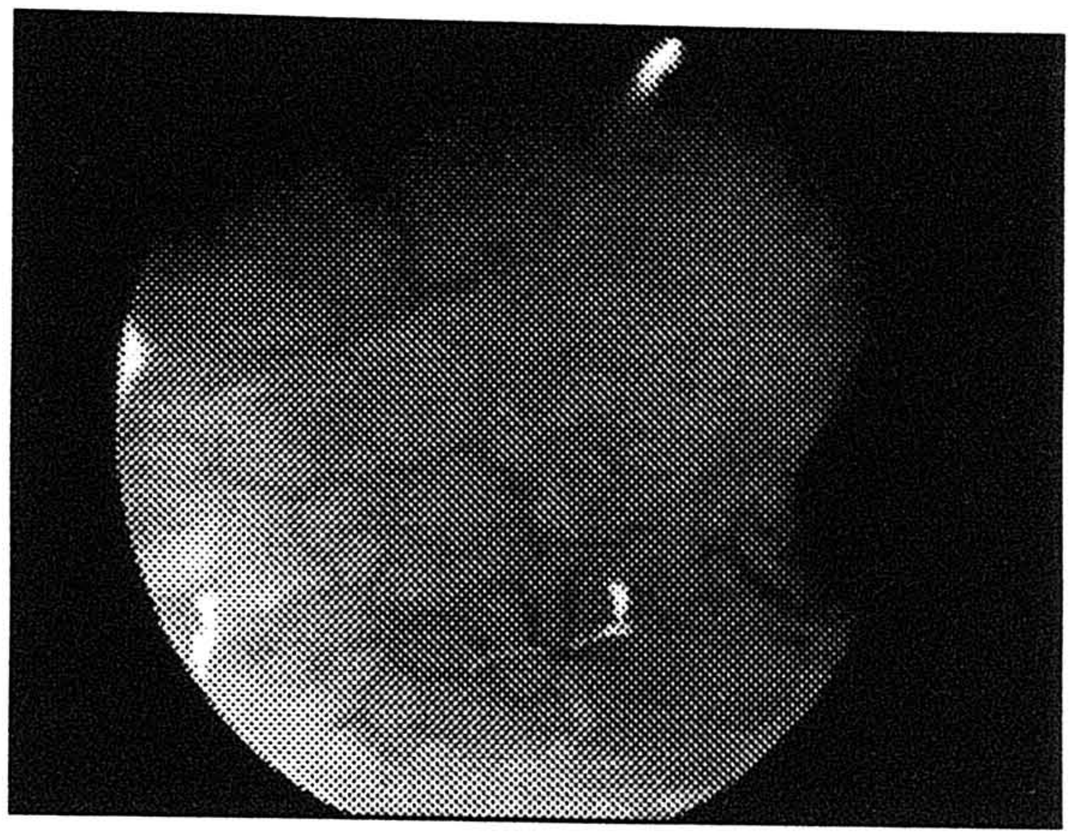


Fig. 10 - Imagem no monitor auxiliar

4.5.2 Botão ACQUIRE

O botão ACQUIRE inicia o processo de aquisição das imagem do videogravador. Uma vez premido o botão e posto o videogravador no modo PLAY é possível ver no monitor auxiliar as imagens do útero previamente gravadas.

No momento em que a imagem for considerada adequada a um posterior processamento, bastará premir uma qualquer tecla do teclado ou um botão do rato. Em seguida é efectuado um varrimento da imagem para eliminação do efeito *fliker* sendo finalmente pedido o nome a dar à imagem então adquirida.

4.5.3 Botão SAVE

Para guardar a imagem no disco recorre-se ao botão SAVE, sendo mais uma vez perguntado o seu *filename*.

4.6 Menu de processamento

O menu de processamento é composto por três botões destinados a seleccionar uma região de interesse na imagem bem como a iniciar o processamento automático ou manual dessa mesma região. Esses botões denominam-se:

- ROI
- AUTO
- MANUAL

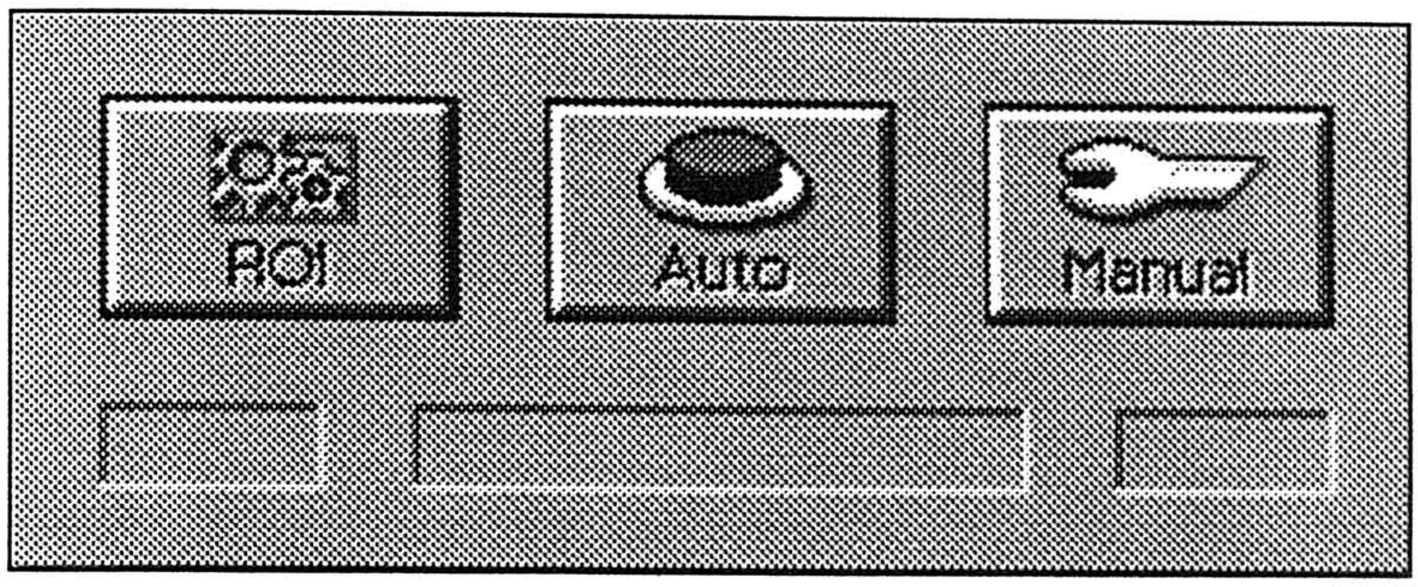


Fig. 11 - Menu de processamento

Quando este menu é acedido surge no monitor auxiliar a região de interesse (ROI) sobreposta à imagem. Caso a sua localização não seja adequada deverá ser seleccionada uma nova região de interesse.

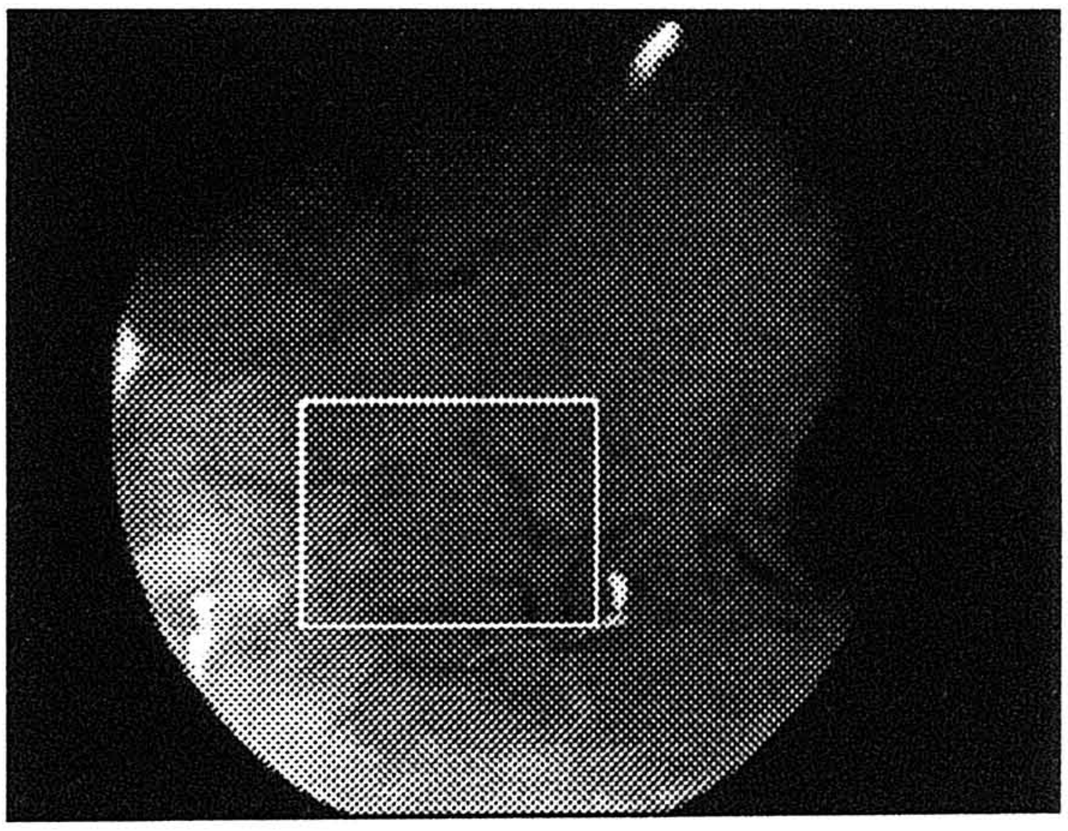


Fig. 12 - Exemplo de uma ROI

4.6.1 Botão ROI

A escolha da região de interesse por intermédio do botão ROI não só permite o processamento de uma parte restrita da imagem, evitando assim zonas onde existam artefactos (brilhos, manchas, etc), como também irá reduzir significativamente os tempos de processamento.

Esta região não deve ser demasiado pequena (largura ou altura inferior a 50 pontos) nem demasiado grande (largura ou altura superior a 250 pontos), caso contrário é dada uma mensagem de erro sendo pedida nova selecção.

Como se pode ver na Fig. 13, no monitor auxiliar surge um cursor que pode ser movimentado com o rato. O utilizador deverá definir a região de interesse escolhendo dois vértices opostos. Para tal desloca o cursor até ao ponto adequado e prime o botão esquerdo do rato. Uma vez seleccionados os dois vértices, a nova ROI é apresentada no monitor. Para auxiliar o processo de escolha da ROI é dada uma indicação no monitor do computador, das coordenadas do cursor, quando da escolha do primeiro vértice, e das dimensões da região, quando da escolha do segundo vértice (Fig. 14).

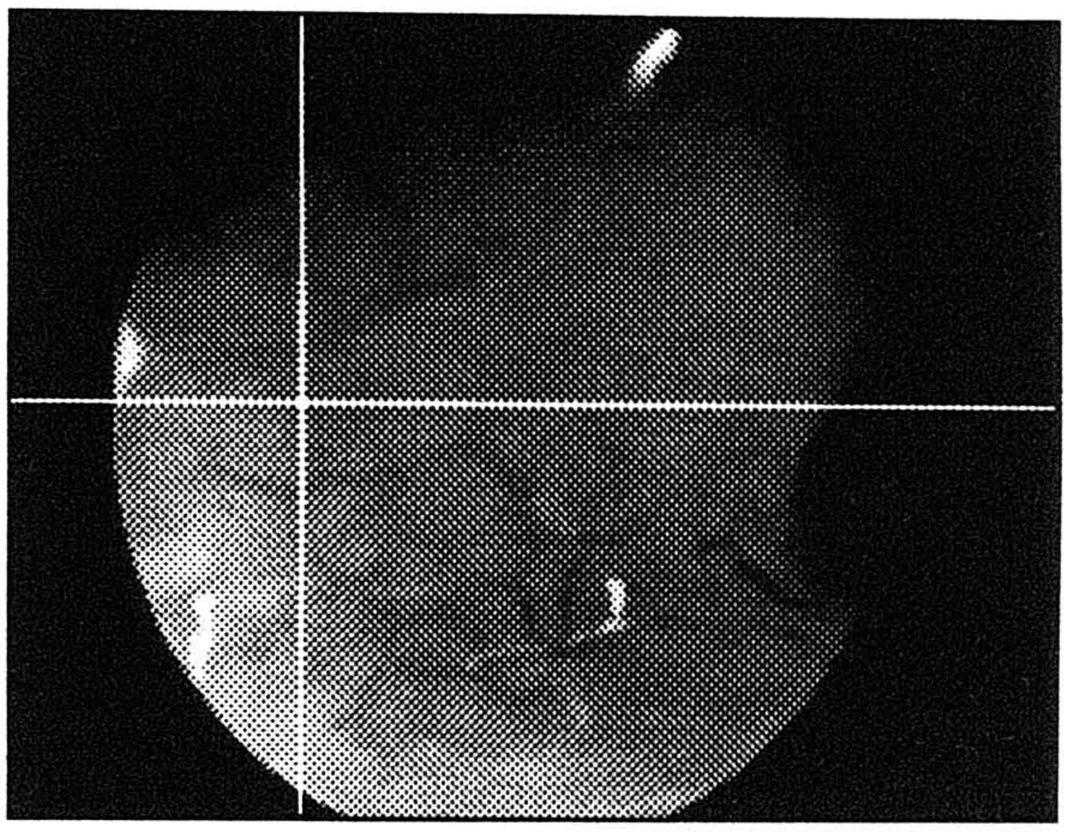


Fig. 13 - Cursor

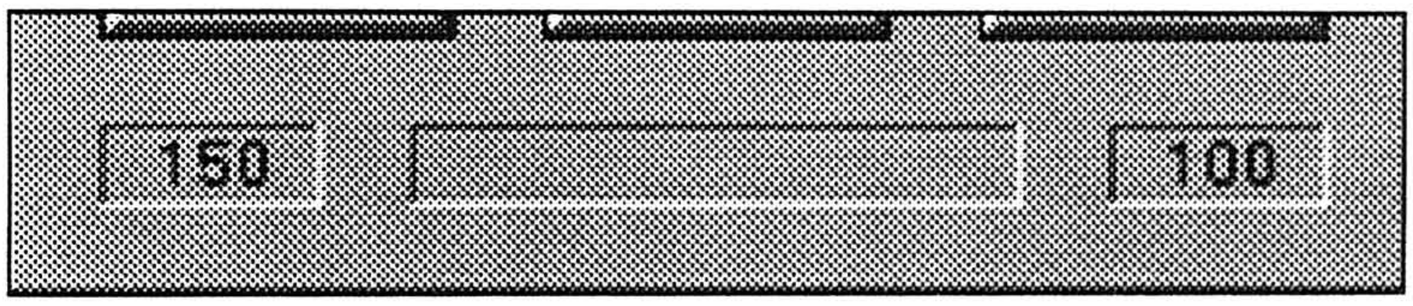


Fig. 14 - Coordenadas do cursor ou dimensões da região

4.6.2 Botão AUTO

O processamento da imagem, ou mais precisamente o processamento da ROI da imagem, é composto pelo seguinte conjunto de operações:

- ***remoção da superfície base*** - elimina as variações de intensidade do fundo da imagem
- ***equalização*** - aumenta o contraste apresentado pela imagem entretanto obtida
- ***detecção de mínimos locais*** - obtem uma imagem de mínimos locais de intensidade
- ***thinning*** - simplifica a imagem anteriormente obtida, adelgaçando-a sem contudo alterar a sua estrutura
- ***decomposição em segmentos*** - decompõem o esqueleto obtido em segmentos
- ***classificação dos segmentos*** - decide quais os segmentos que pelas suas características de intensidade e contraste são classificados como vasos
- ***reconstrução da rede*** - reconstroi a rede vascular a partir destes segmentos
- ***extracção de características*** - determina o valor das características da rede

Duas destas operações têm como variáveis de entrada, mínimos de contraste admissíveis. Por exemplo, a operação de *detecção de mínimos locais* obtém a imagem de mínimos locais escolhendo os pontos que possuem um contraste não inferior a um determinado valor *mpc* (minimum pixel contrast). Por sua vez a operação de *classificação dos segmentos* classifica como vaso todo o segmento com intensidade menor do que 60 e contraste não inferior a *msc* (minimum segment contrast).

No modo de processamento automático, iniciado através do botão AUTO, a escolha dos níveis *mpc* e *msc* é feita pelo programa. Após o arranque do sistema estes valores são iguais a 5 e 20 respectivamente. Em qualquer outra situação os valores mantêm-se iguais aos utilizados no último processamento.

Durante o processamento é indicado o nome da operação a decorrer. A Fig. 15 apresenta um exemplo deste procedimento, quando da execução da operação de *thinning* (uma das mais demoradas).

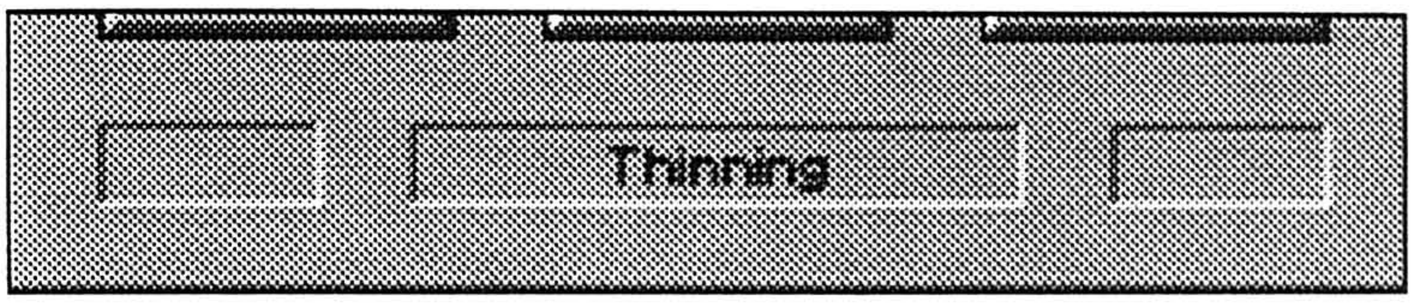


Fig. 15 - Nome da operação a decorrer

Nos quadrantes do monitor auxiliar é possível visualizar os resultados de algumas das operações de processamento. Assim, na Fig. 16 temos:

a - imagem original
b - imagem equalizada
c - segmentos classificados como vasos
d - rede vascular reconstruída

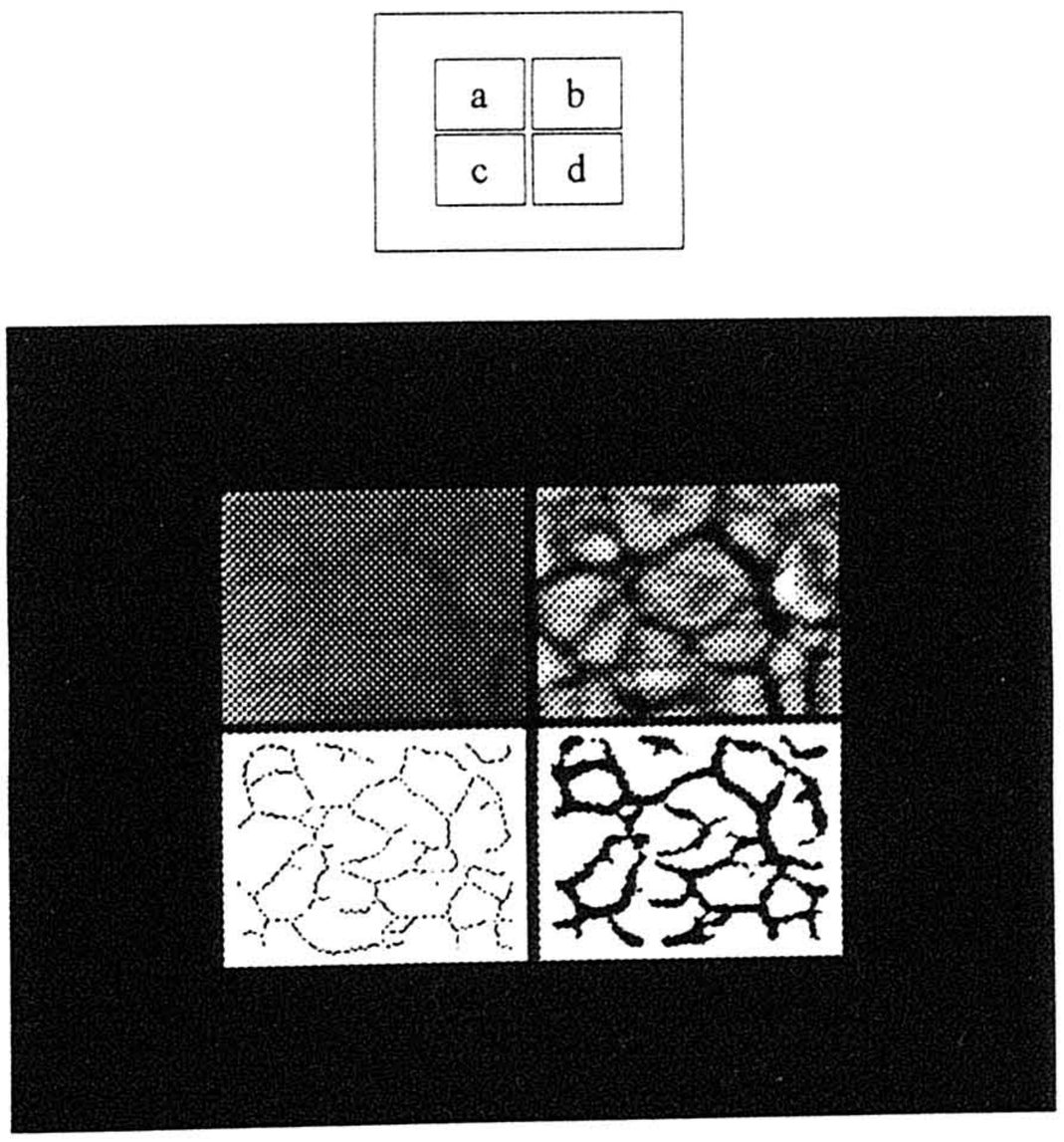


Fig. 16 - Resultados de algumas das operações de processamento

4.6.3 Botão MANUAL

Dada a grande variabilidade de contraste apresentada pelas imagens é muitas vezes preferível a escolha dos níveis de contraste por parte do utilizador. O botão MANUAL dá essa possibilidade uma vez que quando o processamento necessita desses níveis o utilizador é interrogado para o efeito.

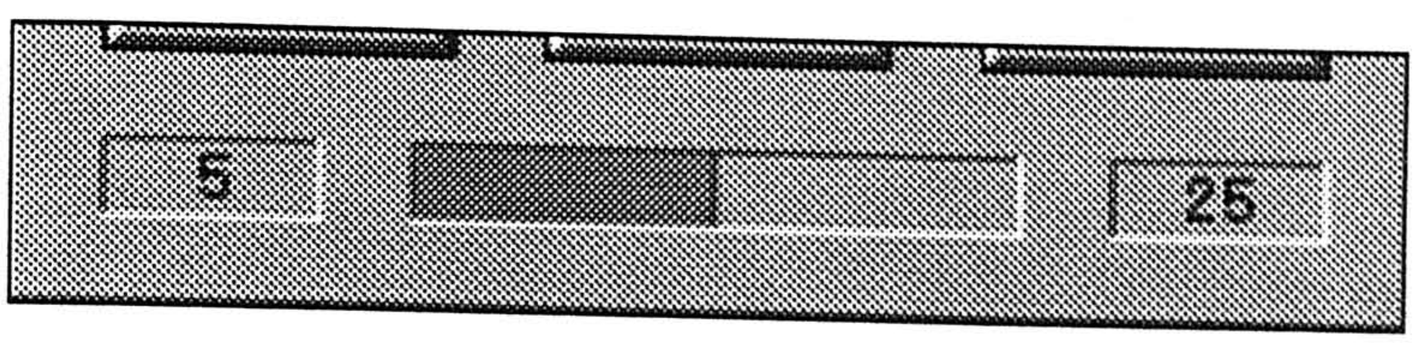


Fig. 17 - Escolha do nível de contraste

Tal como a Fig. 17 mostra, os níveis de contraste são introduzidos por intermédio de uma barra controlada por movimentos laterais do rato. De notar a indicação do valor utilizado no último processamento e o actual. No monitor auxiliar é possível ver, em tempo real, qual o resultado obtido com o actual nível de contraste. Uma vez encontrado o nível mais adequado à imagem a processar basta premir o botão esquerdo do rato por forma a prosseguir o processamento.

Na escolha do primeiro nível de contraste (*mpc*) pretende-se obter um conjunto de pontos que contenham a totalidade da estrutura da rede vascular. Como tal o valor a escolher deverá ser baixo. Isto pode conduzir à escolha de pontos não pertencentes a essa mesma rede. Este facto não é contudo grave uma vez que escolhendo para segundo nível de contraste (*msc*) um valor mais elevado são eliminados todos os segmentos indesejáveis.

4.6.4 Características

No final do processamento é efectuada a análise da rede vascular. Os resultados desta análise são apresentados no monitor através da lista de características (Fig. 18). São também mostrados um conjunto de dados adicionais.

Esses dados adicionais incluem o nome da imagem, hora e data de processamento e ainda a localização e tamanho da região de interesse.

- X - coordenada x do vértice superior esquerdo da imagem
- Y - coordenada y do mesmo vértice
- W - largura da ROI
- H - altura da ROI

São também apresentados os dois níveis de contraste utilizados no processamento da imagem.

- MPC - minimum pixel contrast
- MSC - minimum segment contrast

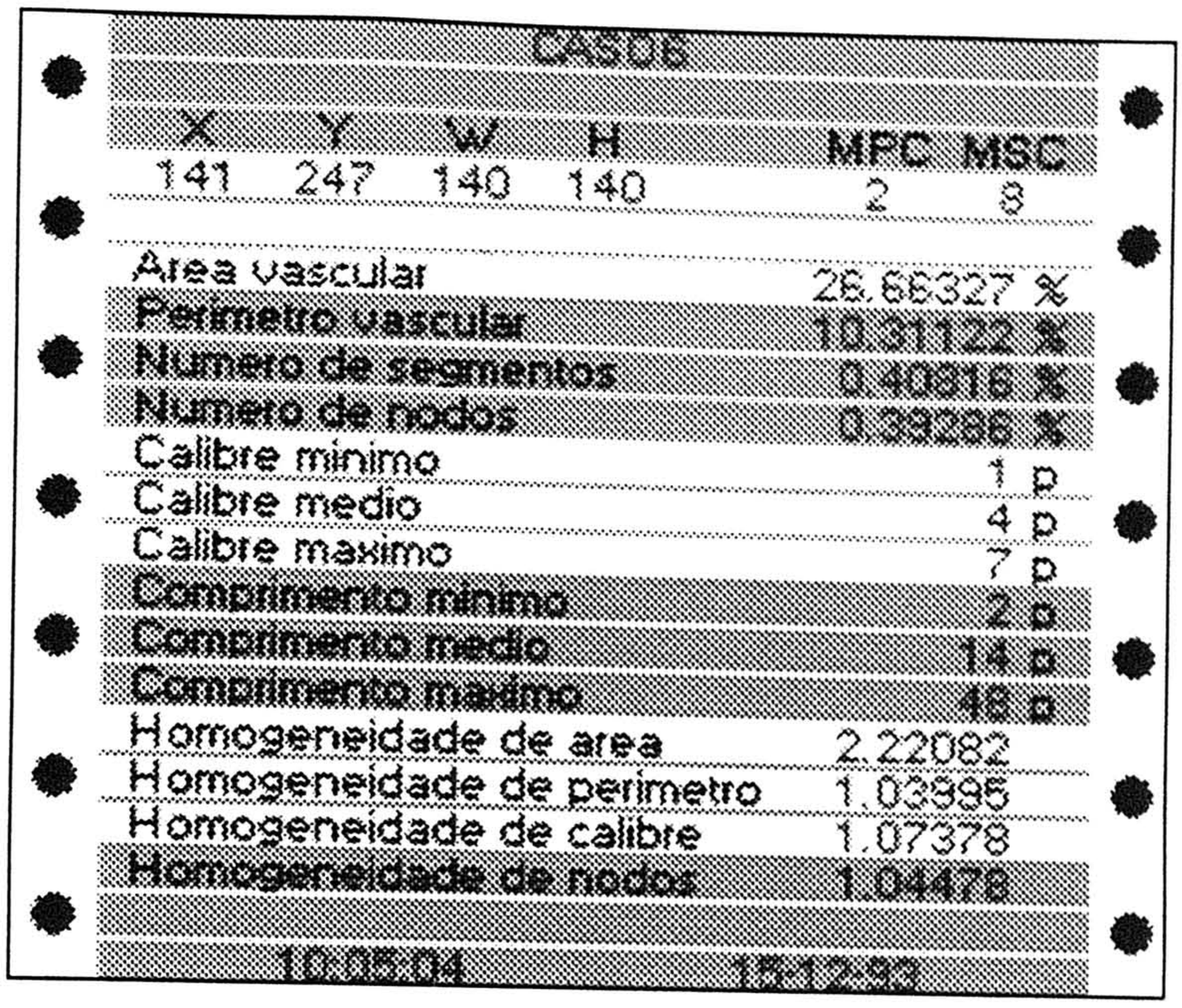


Fig. 18. Características da rede vascular

Da análise efectuada à rede vascular uterina resulta o seguinte conjunto de características:

- AREA - área vascular
- PERI - perímetro vascular
- NSEG - número de segmentos
- NNOD - número de nodos
- CMIN - calibre mínimo
- CMED - calibre médio
- CMAX - calibre máximo
- LMIN - comprimento mínimo
- LMED - comprimento médio
- LMAX - comprimento máximo
- HARE - homogeneidade de área
- HPER - homogeneidade de perímetro
- HCAL - homogeneidade de calibre
- HNOD - homogeneidade de nodos

Os valores das quatro primeiras características são fornecidos em termos percentuais da região de interesse por forma a serem independentes da sua dimensão. Note-se que o número de segmentos não é necessariamente igual ao número de vasos da rede vascular pois estes podem estar fraccionados em vários segmentos. Contudo esta característica assim como o número de nodos (pontos de afluência de três ou mais segmentos) dão uma ideia da complexidade apresentada pela rede. Por fim as características de homogeneidade dão uma informação da uniformidade de distribuição de um dado parâmetro pela ROI.

Após a apresentação dos resultados no monitor o utilizador só terá que premir uma tecla ou botão do rato para aceder ao menu de resultados.

4.7 Menu de resultados

O menu de resultados põe à disposição os botões que possibilitam o envio dos resultados para um disco ou para a impressora. Esses botões são:

- NAME
- DISK
- PRINTER

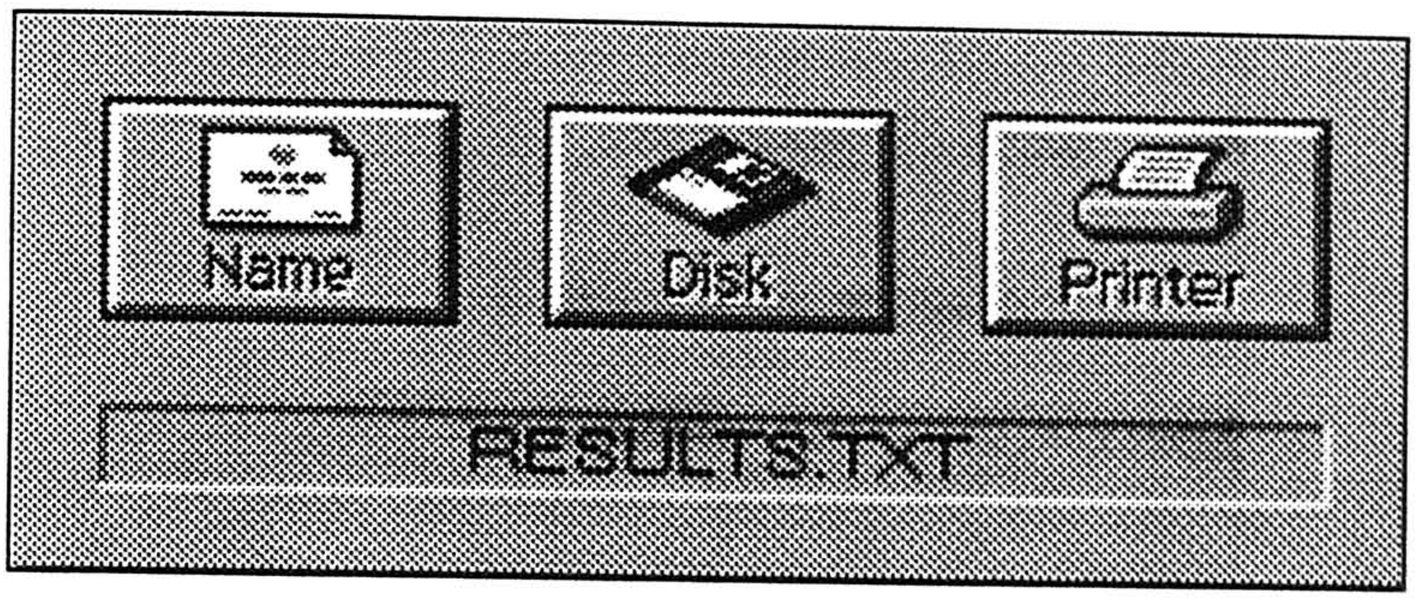


Fig. 19. Menu de resultados

O envio dos resultados para um disco ou para a impressora não é obrigatória. Se o utilizador não os quiser guardar só terá que aceder a um outro menu (de imagem ou de processamento) por intermédio dos botões correspondentes no menu principal.

A selecção de diferentes regiões de interesse e/ou níveis de contraste pode conduzir à obtenção de diferentes resultados no processamento de uma mesma imagem. É contudo possível guardar todos esses resultados para futura análise.

4.7.1 Botão NAME

Este botão permite alterar a nome e a localização do ficheiro onde serão guardados os resultados. Por defeito o ficheiro designa-se RESULTS.TXT e situa-se no directório *spimut*. O processo de introdução do *filename* é semelhante ao mencionado na secção 4.5.1.

4.7.2 Botão DISK

O botão DISK inicia o processo de envio dos resultados para o ficheiro. Caso o ficheiro não exista este é imediatamente criado com um registo contendo a denominação dos campos.

Cada registo de resultados ocupa um total de 158 bytes terminando com os caracteres 13 e 10 (Carriage Return e Line Feed). Os registos são compostos por 23

campos delimitados pelo caracter 9 (Tab). Na tabela seguinte estão indicados os vários campos através da respectiva denominação e tamanho.

DENOMINAÇÃO	TAMANHO*
FILENAME	12
AREA	8
PERI	8
NSEG	8
NNOD	8
CMIN	4
CMED	4
CMAX	4
LMIN	4
LMED	4
LMAX	4
HARE	8
HPERI	8
HCAL	8
HNOD	8
X	3
Y	3
W	3
H	3
MPC	3
MSC	3
TIME	8
DATE	8

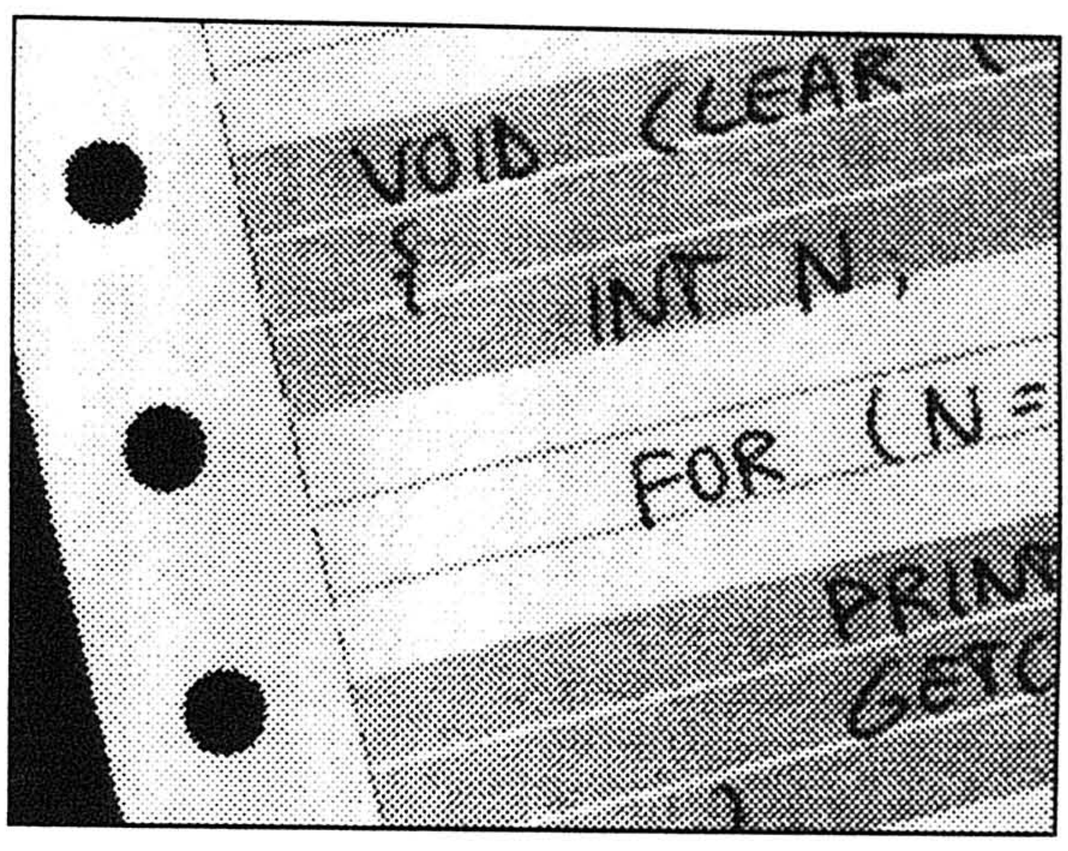
* não incluindo o caracter de delimitação

4.7.3 Botão PRINTER

O botão PRINTER envia os resultados para a impressora. A Fig. 20 mostra como as várias características se distribuem pelo papel.

```
**************************************************
*                                                *
*  CASO6                  10:05:04    15-12-93   *
*                                                *
*              X             141                 *
*              Y             247                 *
*              W             140                 *
*              H             140                 *
*                                                *
*              MPC             2                 *
*              MSC             8                 *
*                                                *
*  Area vascular                      26.66327 %  *
*  Perimetro vascular                 10.31122 %  *
*  Numero de segmentos                 0.40816 %  *
*  Numero de nodos                     0.39286 %  *
*  Calibre minimo                            1 p  *
*  Calibre medio                             4 p  *
*  Calibre maximo                            7 p  *
*  Comprimento minimo                        2 p  *
*  Comprimento medio                        14 p  *
*  Comprimento maximo                       48 p  *
*  Homogeneidade de area              2.22082     *
*  Homogeneidade de perimetro         1.03995     *
*  Homogeneidade de area              1.07378     *
*  Homogeneidade de area              1.04478     *
*                                                *
**************************************************
```

Fig. 20. Resultados impressos pela impressora



APÊNDICE B

LISTAGENS DO PROGRAMA

Neste apêndice são fornecidas as listagens do código fonte de todos os ficheiros que compõem o programa de processamento e análise de imagens histeroscópicas uterinas (SPIMUT).

spimut.h

```
#define LEFT        0
#define CENTER      1
#define RIGHT       2

#define BLACK       0
#define WHITE      15
#define LIGHTGRAY   7
#define DARKGRAY    8
#define LIGHTRED   12
#define RED         4

#define FALSE       0
#define TRUE        1

typedef struct {
  int x;
  int y;
} region;

typedef unsigned char PIXEL;
typedef PIXEL* VECTOR;
typedef VECTOR*  MATRIX;

typedef struct {
  MATRIX matrix;
  int xsize;
  int ysize;
} IMAGE;

typedef struct {

  char  name[13];

  float area;
  float peri;
  float nseg;
  float nnod;

  int   cmin;
  int   cmed;
  int   cmax;
  int   lmin;
  int   lmed;
  int   lmax;

  float hare;
  float hper;
  float hdia;
  float hnod;

  int   x;
  int   y;
  int   w;
  int   h;

  int   thrs;
  int   cont;

  char  time[9];
  char  date[9];

} results;
```

mouse.h

```
int  MouseInstalled(void);
void MouseReset(int *NumButtons);
void MouseShowPointer(void);
void MouseHidePointer(void);
void MouseGetButtons(int *ButtonStatus, int *PointerCol,
  int *PointerRow);
void MouseSetPointer(int PointerCol, int PointerRow);
void MouseGetButtonPress(int Button, int *ButtonStatus,
  int *NumberPresses, int *PointerCol, int *PointerRow);
void MouseGetButtonRelease(int Button, int *ButtonStatus,
  int *NumberReleases, int *PointerCol, int *PointerRow);
void MouseSetPointerHorizArea(int MinCol, int MaxCol);
void MouseSetPointerVertArea(int MinRow, int MaxRow);
void MouseSetGraphPointer(int HotSpotHoriz, int HotSpotVert,
  unsigned far *FarPtrMasks);
void MouseSetTextPointer(int PointerType, int SMaskScanStart,
  int PMaskScanStop);
void MouseGetMickeys(int *HorizCount, int *VertCount);
void MouseSetIntHandler(int CallMask,
  void (far *FarPtrHandler)(void));
void MouseSetRatio(int HorizRatio, int VertRatio);
void MouseGetStorage(int *BufferSize);
void MouseSaveState(unsigned char far *FarPtrBuffer);
void MouseRestoreState(unsigned char far *FarPtrBuffer);
```

mouse.c

```
#include <dos.h>

int MouseInstalled(void)
{
  unsigned char far *FarPtrMouDriver;

  FarPtrMouDriver = (unsigned char far *) _dos_getvect(0x33);
  if (FarPtrMouDriver == 0 || *FarPtrMouDriver == 0xcf)
  return(0);
  else
  return(1);
}

void MouseReset(int *NumButtons)
{
  union REGS Reg;

  Reg.x.ax = 0;
  int86(0x33, &Reg, &Reg);
  *NumButtons = Reg.x.bx;
}

void MouseShowPointer(void)
{
  union REGS Reg;

  Reg.x.ax = 1;
  int86(0x33, &Reg, &Reg);
}

void MouseHidePointer(void)
{
  union REGS Reg;

  Reg.x.ax = 2;
  int86(0x33, &Reg, &Reg);
}

void MouseGetButtons(int *ButtonStatus, int *PointerCol,
  int *PointerRow)
{
  union REGS Reg;

  Reg.x.ax = 3;
  int86(0x33, &Reg, &Reg);
  *ButtonStatus = Reg.x.bx;
  *PointerCol = Reg.x.cx;
  *PointerRow = Reg.x.dx;
}

void MouseSetPointer(int PointerCol, int PointerRow)
{
  union REGS Reg;

  Reg.x.ax = 4;
  Reg.x.cx = PointerCol;
  Reg.x.dx = PointerRow;
  int86(0x33, &Reg, &Reg);
}

void MouseGetButtonPress(int Button, int *ButtonStatus,
  int *NumberPresses, int *PointerCol, int *PointerRow)
{
  union REGS Reg;

  Reg.x.ax = 5;
  Reg.x.bx = Button;
  int86(0x33, &Reg, &Reg);
  *ButtonStatus = Reg.x.ax;
  *NumberPresses = Reg.x.bx;
  *PointerCol = Reg.x.cx;
  *PointerRow = Reg.x.dx;
}

void MouseGetButtonRelease(int Button, int *ButtonStatus,
  int *NumberReleases, int *PointerCol, int *PointerRow)
{
  union REGS Reg;

  Reg.x.ax = 6;
  Reg.x.bx = Button;
  int86(0x33, &Reg, &Reg);
  *ButtonStatus = Reg.x.ax;
  *NumberReleases = Reg.x.bx;
  *PointerCol = Reg.x.cx;
  *PointerRow = Reg.x.dx;
}

void MouseSetPointerHorizArea(int MinCol, int MaxCol)
{
  union REGS Reg;

  Reg.x.ax = 7;
  Reg.x.cx = MinCol;
  Reg.x.dx = MaxCol;
  int86(0x33, &Reg, &Reg);
}

void MouseSetPointerVertArea(int MinRow, int MaxRow)
{
  union REGS Reg;

  Reg.x.ax = 8;
  Reg.x.cx = MinRow;
  Reg.x.dx = MaxRow;
  int86(0x33, &Reg, &Reg);
}

void MouseSetGraphPointer(int HotSpotHoriz, int HotSpotVert,
          unsigned far *FarPtrMasks)
{
  union REGS Reg;
  struct SREGS SReg;

  Reg.x.ax = 9;
  Reg.x.bx = HotSpotHoriz;
  Reg.x.cx = HotSpotVert;
  Reg.x.dx = FP_OFF(FarPtrMasks);
  SReg.es = FP_SEG(FarPtrMasks);
  int86x(0x33, &Reg, &Reg, &SReg);
}
```

```
void MouseSetTextPointer(int PointerType, int SMaskScanStart,
        int PMaskScanStop)
{
  union REGS Reg;

  Reg.x.ax = 10;
  Reg.x.bx = PointerType;
  Reg.x.cx = SMaskScanStart;
  Reg.x.dx = PMaskScanStop;
  int86(0x33, &Reg, &Reg);
}

void MouseGetMickeys(int *HorizCount, int *VertCount)
{
  union REGS Reg;

  Reg.x.ax = 11;
  int86(0x33, &Reg, &Reg);
  *HorizCount = Reg.x.cx;
  *VertCount = Reg.x.dx;
}

void MouseSetIntHandler(int CallMask,
  void (far *FarPtrHandler)(void))
{
  union REGS Reg;
  struct SREGS SReg;

  Reg.x.ax = 12;
  Reg.x.cx = CallMask;
  Reg.x.dx = FP_OFF(FarPtrHandler);
  SReg.es = FP_SEG(FarPtrHandler);
  int86x(0x33, &Reg, &Reg, &SReg);
}

void MouseSetRatio(int HorizRatio, int VertRatio)
{
  union REGS Reg;

  Reg.x.ax = 15;
  Reg.x.cx = HorizRatio;
  Reg.x.dx = VertRatio;
  int86(0x33, &Reg, &Reg);
}

void MouseGetStorage(int *BufferSize)
{
  union REGS Reg;

  Reg.x.ax = 21;
  int86(0x33, &Reg, &Reg);
  *BufferSize = Reg.x.bx;
}

void MouseSaveState(unsigned char far *FarPtrBuffer)
{
  union REGS Reg;
  struct SREGS SReg;

  Reg.x.ax = 22;
  Reg.x.dx = FP_OFF(FarPtrBuffer);
  SReg.es = FP_SEG(FarPtrBuffer);
  int86x(0x33, &Reg, &Reg, &SReg);
}

void MouseRestoreState(unsigned char far *FarPtrBuffer)
{
  union REGS Reg;
  struct SREGS SReg;

  Reg.x.ax = 23;
  Reg.x.dx = FP_OFF(FarPtrBuffer);
  SReg.es = FP_SEG(FarPtrBuffer);
  int86x(0x33, &Reg, &Reg, &SReg);
}
```

main.c

```
#include <graph.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dos.h>

#include "mouse.h"
#include "spimut.h"

void change_regions (void);
void preenche_array (void);
void error (void);

void *buttons;

char filename[50];

char fic_name[50];

region SIZE;
region ROI;
region REG_A;
region REG_B;
region REG_C;
region REG_D;

int thrs;
int cmin;

results res;

void far hhandler (unsigned deverr, unsigned doserr,
  unsigned far *hdr )
{
  error ();

  _hardretn ( doserr);
}

void init_dtiris (void)
{
  IS_ERROR_MESSAGES(0);

  if(IS_INITIALIZE ()) {
    printf("Error: DT2851 not present\n");
    exit(1);
  }

  IS_RESET ();
  IS_SET_SYNC_SOURCE (0);

  IS_FRAME_CLEAR (0);
  IS_FRAME_CLEAR (1);

  IS_SELECT_INPUT_FRAME (1);
  IS_SELECT_OUTPUT_FRAME (1);

  IS_SELECT_ILUT (0);
  IS_SELECT_OLUT (0);

  IS_SET_ACTIVE_REGION (0, 0, 512, 512);

  IS_DISPLAY (1);
}

void init_mouse (void)
{
  int NumButtons;

  if( !MouseInstalled() ) {
    printf("Error: Mouse not present\n");
    exit(1);
  }

  MouseReset(&NumButtons);
}

void load_pics (void)
{
  FILE *in;

  if ((in = fopen("pics.dat", "rb")) == NULL) {
    printf ("Error: Can't register pics\n");
    exit(1);
  }

  if ((buttons = malloc((unsigned) 38436)) == NULL) {
    printf("Error: Allocation memory\n");
    exit(1);
  }

  fread (buttons, (unsigned) 38436, 1, in);

  fclose (in);
}

void init_graph (void)
{
  if( _registerfonts( "*.FON" ) <= 0 ) {
    _setvideomode (_DEFAULTMODE);
    printf( "Error: Can't register fonts\n" );
    exit( 1 );
  }

  if ( !_setvideomode( _VRES16COLOR ) ) {
    printf("Error: VGA not present\n");
    exit( 1 );
  }

  _setfont( "t'helv' h12" );
}

void init_var (void)
{
  strcpy (filename, "untitled");

  strcpy (fic_name, "results.txt");

  ROI.x  = 200;
  ROI.y  = 200;
  SIZE.x = 200;
  SIZE.y = 200;

  change_regions ();

  thrs = 5;
  cmin = 20;

  preenche_array ();
}

void end_dtiris (void)
{
  IS_FRAME_CLEAR(0);
  IS_FRAME_CLEAR(1);
  IS_END();
}

void end_graph (void)
{
  _unregisterfonts();
}
```

```
  _setvideomode (_DEFAULTMODE);
}

void main (void)
{
  _harderr ( hhandler );

  init_dtiris ();

  init_mouse ();

  load_pics ();

  init_graph ();

  init_var ();

  main_menu ();

  end_dtiris ();

  end_graph ();
}
```

ackn.c

```
#include <graph.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>
#include <string.h>

#include "mouse.h"
#include "spimut.h"

extern void *buttons;

int print (int x1, int y, int just, int color, char *str);

void scroll (void)
{
  int n;
  PIXEL buf[250];

  for (n=37; n<=103; n++) {

    _getimage (209, n, 430, n, buf);
    _putimage (209, n-1, buf, _GPSET);
  }
}

void scroll_spimut (void)
{
  int n;
  PIXEL buf[250];
  int *p;

  p = (int *) buf;

  *p++ = 222;
  *p   =   1;

  for (n=0; n<68; n++) {

    memcpy (buf+4, (char*)buttons+30816+4+n*112, 112);

    _putimage (209, 103, buf, _GPSET);
    scroll ();
  }
}

void scrolling (int lila)
{
  int m, n;
  int from_message, to_message;

  struct {
    char *str;
    int color;
  } mess[16] = {
    { "", WHITE },
    { "Author", WHITE },
    { "Helio Mendonca", LIGHTRED },
    { "", WHITE },
    { "Technical Advisor", WHITE },
    { "Prof. Marques de Sa", LIGHTRED },
    { "", WHITE },
    { "Medical Advisor", WHITE },
    { "Dr. Silva Carvalho", LIGHTRED },
    { "", WHITE },
    { "Sponsor", WHITE },
    { "JNICT", LIGHTRED },
    { "", WHITE },
    { "Dedicated to", WHITE },
    { "Marilia Claudia", LIGHTRED },
    { "", WHITE }
  };

  wait_release ();

  MouseHidePointer ();

  from_message = (lila) ? 12 :  0;
  to_message   = (lila) ? 16 : 13;

  for (m=from_message; m<to_message; m++) {

    for (n=0; n<14; n++) {

      _setcolor (BLACK);
      _rectangle (_GFILLINTERIOR, 209, 103, 430, 103);

      _setcliprgn (209, 103, 430, 103);

      print (320, 103-n, CENTER, mess[m].color, mess[m].str);

      _setcliprgn (0, 0, 639, 479);

      scroll ();
    }
  }

  scroll_spimut ();

  MouseShowPointer ();
}

void abou_proc (void)
{
  scrolling (FALSE);
}

void lila_proc (void)
{
  scrolling (TRUE);
}
```

menu.c

```
#include <graph.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <bios.h>

#include "mouse.h"
#include "spimut.h"

#define UP          0
#define DOWN        1

#define IMAG_BUT    0
#define PROC_BUT    1
#define EXIT_BUT    2
#define LOAD_BUT    3
#define ACQU_BUT    4
#define SAVE_BUT    5
#define ROIN_BUT    6
#define AUTO_BUT    7
#define STEP_BUT    8
#define NAME_BUT    9
#define DISK_BUT   10
#define PRNT_BUT   11
#define ABOU_BUT   12
#define LILA_BUT   13

extern void *buttons;
extern char filename[50];
extern char fic_name[50];

void load_proc (void);
void acqu_proc (void);
void save_proc (void);
void roin_proc (void);
void auto_proc (void);
void step_proc (void);
void name_proc (void);
void disk_proc (void);
void prnt_proc (void);
void abou_proc (void);
void lila_proc (void);

int print (int x1, int y, int just, int color, char *str);
void draw_reg (void);
void error (void);
void get_filename (char *out, char *in);

#define N_MAIN 3
#define N_IMAG 3
#define N_PROC 3
#define N_RESL 3

typedef struct {
  int desg;
  int x1, y1, x2, y2;
} but_info;

but_info abou_but[2] = {
  { LILA_BUT, 308, 214, 315, 221 },
  { ABOU_BUT, 208,  35, 431, 104 }
};

but_info main_but [N_MAIN] = {
  { IMAG_BUT, 208, 155, 271, 194 },
  { PROC_BUT, 288, 155, 351, 194 },
  { EXIT_BUT, 368, 155, 431, 194 }
};

but_info imag_but [N_IMAG] = {
  { LOAD_BUT, 208, 275, 271, 314 },
  { ACQU_BUT, 288, 275, 351, 314 },
  { SAVE_BUT, 368, 275, 431, 314 }
};
```

```
but_info proc_but [N_PROC] = {
  { ROIN_BUT, 208, 275, 271, 314 },
  { AUTO_BUT, 288, 275, 351, 314 },
  { STEP_BUT, 368, 275, 431, 314 }
};

but_info resl_but [N_PROC] = {
  { NAME_BUT, 208, 275, 271, 314 },
  { DISK_BUT, 288, 275, 351, 314 },
  { PRNT_BUT, 368, 275, 431, 314 }
};

int imag_flag;
int proc_flag;
int resl_flag;

void wait_release(void)
{
  int Press, x, y;

  do {
    MouseGetButtons(&Press, &x, &y);
  } while (Press);
}

void wait_press(void)
{
  int Press, x, y;

  do {
    MouseGetButtons(&Press, &x, &y);
  } while (!Press);
}

void draw_bar (int x1, int y1, int x2, int y2, int type)
{
  _setcolor (LIGHTGRAY);
  _rectangle( _GFILLINTERIOR, x1, y1, x2, y2);

  if (type == UP) _setcolor (WHITE);
  else    _setcolor (DARKGRAY);

  _moveto (x1, y2);
  _lineto (x1, y1);
  _lineto (x2, y1);

  if (type == UP) _setcolor (DARKGRAY);
  else    _setcolor (WHITE);

  _lineto (x2, y2);
  _lineto (x1, y2);
}

void activate_but (int x, int y, void *p)
{
  MouseHidePointer ();
  _putimage (x, y, (char*)p+1284, _GPSET);
  MouseShowPointer ();

  wait_release ();

  MouseHidePointer ();
  _putimage (x, y, p, _GPSET);
  MouseShowPointer ();
}

int in_box (int x, int y, but_info *p, int N)
{
  int box;
  int *q;

  for (box=0; box<N; box++) {

    q = (int *) (p + box) + 1;

    if ((x > *q++) && (y > *q++) && (x < *q++) && (y < *q))
      return ((*(p+box)).desg);
  }

  return (-1);
}

void clear_level (void)
{
  _setcolor (BLACK);
  _rectangle (_GFILLINTERIOR, 192, 260, 447, 479);
}

void resl_wait (int PointerCol, int PointerRow)
{
  int box;

  box = in_box (PointerCol, PointerRow, resl_but, N_RESL);

  switch (box) {

    case NAME_BUT: {
      activate_but (resl_but[0].x1, resl_but[0].y1,
        (char*)buttons+resl_but[0].desg*1284*2);
      MouseHidePointer ();
      name_proc ();
      MouseShowPointer ();
      break;
    }

    case DISK_BUT: {
      activate_but (resl_but[1].x1, resl_but[1].y1,
        (char*)buttons+resl_but[1].desg*1284*2);
      MouseHidePointer ();
      disk_proc ();
      MouseShowPointer ();
      break;
    }

    case PRNT_BUT: {
      activate_but (resl_but[2].x1, resl_but[2].y1,
        (char*)buttons+resl_but[2].desg*1284*2);
      MouseHidePointer ();
      prnt_proc ();
      MouseShowPointer ();
      break;
    }
  }
}

void resl_menu (void)
{
  int n;
  char str[15];

  draw_bar (192, 260, 447, 359, UP);
  draw_bar (208, 330, 431, 344, DOWN);

  for (n=0; n<N_RESL; n++)
    _putimage (resl_but[n].x1, resl_but[n].y1,
      (char*)buttons+resl_but[n].desg*1284*2, _GPSET);

  get_filename (str, fic_name);
  print (320, 331, CENTER, BLACK, str);
}

void imag_wait (int PointerCol, int PointerRow)
{
  int box;

  box = in_box (PointerCol, PointerRow, imag_but, N_IMAG);

  switch (box) {

    case LOAD_BUT: {
      activate_but (imag_but[0].x1, imag_but[0].y1,
        (char*)buttons+imag_but[0].desg*1284*2);
      load_proc ();
      break;
    }

    case ACQU_BUT: {
      activate_but (imag_but[1].x1, imag_but[1].y1,
        (char*)buttons+imag_but[1].desg*1284*2);
      acqu_proc ();
      break;
    }

    case SAVE_BUT: {
      activate_but (imag_but[2].x1, imag_but[2].y1,
        (char*)buttons+imag_but[2].desg*1284*2);
      save_proc ();
      break;
    }
  }
}

void imag_menu (void)
{
  int n;
  char str[50];

  draw_bar (192, 260, 447, 359, UP);
  draw_bar (208, 330, 431, 344, DOWN);

  for (n=0; n<N_IMAG; n++)
    _putimage (imag_but[n].x1, imag_but[n].y1,
      (char*)buttons+imag_but[n].desg*1284*2, _GPSET);

  IS_DISPLAY (0);
  IS_FRAME_COPY (0, 1);
  IS_SELECT_OLUT (0);
  IS_DISPLAY (1);

  get_filename (str, filename);
  print (320, 331, CENTER, BLACK, str);
}

void proc_wait (int PointerCol, int PointerRow)
{
  int box;

  box = in_box (PointerCol, PointerRow, proc_but, N_PROC);

  switch (box) {

    case ROIN_BUT: {
      activate_but (proc_but[0].x1, proc_but[0].y1,
        (char*)buttons+proc_but[0].desg*1284*2);
      roin_proc ();
      break;
    }

    case AUTO_BUT: {
      activate_but (proc_but[1].x1, proc_but[1].y1,
        (char*)buttons+proc_but[1].desg*1284*2);
      auto_proc ();
      MouseHidePointer ();
      clear_level ();
      resl_menu ();
      MouseShowPointer ();
      proc_flag = 0;
      resl_flag = 1;
      break;
    }

    case STEP_BUT: {
      activate_but (proc_but[2].x1, proc_but[2].y1,
```

```
        (char*)buttons+proc_but[2].desg*1284*2);
      step_proc ();
      MouseHidePointer ();
      clear_level ();
      resl_menu ();
      MouseShowPointer ();
      proc_flag = 0;
      resl_flag = 1;
      break;
    }
  }
}

void proc_menu (void)
{
  int n;

  draw_bar (192, 260, 447, 359, UP);
  draw_bar (208, 330, 247, 344, DOWN);
  draw_bar (264, 330, 375, 344, DOWN);
  draw_bar (392, 330, 431, 344, DOWN);

  for (n=0; n<N_PROC; n++)
    _putimage (proc_but[n].x1, proc_but[n].y1,
      (char*)buttons+proc_but[n].desg*1284*2, _GPSET);

  IS_DISPLAY (0);
  IS_SET_ACTIVE_REGION (0, 0, 512, 512);
  IS_FRAME_COPY (0, 1);
  IS_PERFORM_FEEDBACK (1, 5);
  IS_SELECT_OLUT (7);
  IS_DISPLAY (1);

  IS_SET_FOREGROUND (255);
  draw_reg ();
}

int main_wait (int PointerCol, int PointerRow)
{
  int quit = FALSE;
  int box;

  box = in_box (PointerCol, PointerRow, main_but, N_MAIN);

  switch (box) {

    case IMAG_BUT: {
      activate_but (main_but[0].x1, main_but[0].y1,
        (char*)buttons+main_but[0].desg*1284*2);
      MouseHidePointer ();
      clear_level ();
      imag_menu ();
      MouseShowPointer ();
      imag_flag = 1;
      proc_flag = 0;
      resl_flag = 0;
      break;
    }

    case PROC_BUT: {
      activate_but (main_but[1].x1, main_but[1].y1,
        (char*)buttons+main_but[1].desg*1284*2);
      MouseHidePointer ();
      clear_level ();
      proc_menu ();
      MouseShowPointer ();
      proc_flag = 1;
      imag_flag = 0;
      resl_flag = 0;
      break;
    }

    case EXIT_BUT: {
      activate_but (main_but[2].x1, main_but[2].y1,
        (char*)buttons+main_but[2].desg*1284*2);
      quit = TRUE;
      break;
    }
  }

  if (resl_flag) resl_wait(PointerCol, PointerRow);
  if (imag_flag) imag_wait(PointerCol, PointerRow);
  if (proc_flag) proc_wait(PointerCol, PointerRow);

  return (quit);
}

void main_menu (void)
{
  int ButtonStatus, PointerCol, PointerRow;
  int quit = FALSE;
  int x, y;
  int n;
  int box;

  imag_flag = 0;
  proc_flag = 0;

  draw_bar (192,  20, 447, 119, UP);
  draw_bar (208,  35, 431, 104, DOWN);

  _putimage (209, 36, (char *)buttons+30816, _GPSET);

  draw_bar (192, 140, 447, 239, UP);
  draw_bar (208, 210, 431, 224, DOWN);
  for (n=0; n<N_MAIN; n++)
    _putimage (main_but[n].x1, main_but[n].y1,
      (char*)buttons+main_but[n].desg*1284*2, _GPSET);

  print (320, 211, CENTER, BLACK, "Helio c 1993");

  MouseShowPointer ();
```

```
  do {
    MouseGetButtons(&ButtonStatus, &PointerCol, &PointerRow);

    if (ButtonStatus) {

      box = in_box (PointerCol, PointerRow, abou_but, 2);

      switch (box) {

        case ABOU_BUT: {
          abou_proc ();
          break;
        }

        case LILA_BUT: {
          lila_proc ();
          break;
        }
      }

      quit = main_wait (PointerCol, PointerRow);
    }

  } while (!quit);

  MouseHidePointer ();

  free (buttons);
}
```

imag.c

```
#include <graph.h>
#include <bios.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>

#include <conio.h>

#include "mouse.h"
#include "spimut.h"

extern char filename[50];

void wait_release (void);

#define MARGIN 2

int print (int x1, int y, int just, int color, char *str)
{
  int w;

  _setcolor (color);

  w = _getgtextextent (str);

  switch (just) {
    case LEFT: {
      _moveto (x1, y);
      _outgtext (str);
      break;
    }
    case CENTER: {
      _moveto (x1-w/2, y);
      _outgtext (str);
      break;
    }
    case RIGHT: {
      _moveto (x1-w+1, y);
      _outgtext (str);
      break;
    }
  }

  return (w);
}

void clear_kb_buf (void)
{
  while ( _bios_keybrd(_KEYBRD_READY) )
    getch ();
}

void special_getch (void)
{
  int ButtonStatus, PointerCol, PointerRow;

  clear_kb_buf ();

  do {

    MouseGetButtons(&ButtonStatus, &PointerCol, &PointerRow);

  } while ( !ButtonStatus && !_bios_keybrd(_KEYBRD_READY) );

  clear_kb_buf ();

  wait_release ();
}

void input (int x1, int y, int x2, char *str)
{
  char c[2] = "a";
  int i;
  int l;
  int x;
```

```
clock_t t;
int cursor;

clear_kb_buf ();

l = strlen(str);

_setcolor (7);
_rectangle( _GFILLINTERIOR, x1, y, x2, y+12);
_moveto (x1+MARGIN, y);
_setcolor (0);
_outgtext (str);

x1 += MARGIN + _getgtextextent (str);

do {

  cursor = 0;
  t = clock();

  _setcolor(DARKGRAY);
  _moveto (x1+2, y+2);
  _lineto (x1+2, y+10);

  while ( !_bios_keybrd(_KEYBRD_READY) ) {

    if ((clock() - t) > 500) {
      t = clock();

      if (cursor) _setcolor(DARKGRAY);
      else        _setcolor(LIGHTGRAY);
      _moveto (x1+2, y+2);
      _lineto (x1+2, y+10);

      cursor = 1 - cursor;
    }
  }

if ((*c = _bios_keybrd(_KEYBRD_READ)) != 13) {

  x = x1;

  if (*c == 8) {
    if (l != 0) {
      x1 -= _getgtextextent (str+l-1);
      _setcolor (7);
      _rectangle( _GFILLINTERIOR, x1, y, x2, y+12);
      str[--l] = '\0';
    }
  }

  if (*c > ' ') {
    _setcolor(LIGHTGRAY);
    _moveto (x+2, y+2);
    _lineto (x+2, y+10);
    x1 += _getgtextextent (c);
    if (x1<x2-MARGIN) {
      str[l] = *c;
      str[++l] = '\0';
      _moveto (x, y);
      _setcolor (0);
      _outgtext (c);
    }
    else x1 = x;
  }
 }
} while (*c != 13);
}

void clear (int x1, int y, int x2)
{
  _setcolor (7);
  _rectangle( _GFILLINTERIOR, x1, y, x2, y+12);
}

void error (void)
{
  clock_t start;

  MouseHidePointer ();

  clear (209, 211, 430);
  print (320, 211, CENTER, RED, "ERROR");

  printf ("%c", 7);

  start = clock();

  while ((clock() - start) < 500);

  clear (209, 211, 430);
  print (320, 211, CENTER, BLACK, "Helio c 1993");

  MouseShowPointer ();
}

void get_filename (char *out, char *in)
{
  char str[50];
  char *p;
  int n;

  strcpy (str, in);

  for (p=str+strlen(str)-1; p>=str; p--)
    if ((*p=='\\') || (*p==':')) break;

  strcpy (out, p+1);

  out[12] = '\0';

  strupr (out);
```

```
}

void load_proc (void)
{
  int w;
  char str[50];

  MouseHidePointer ();

  clear (209, 331, 430);

  w = print (210, 331, LEFT, BLACK, "Filename:");

  input (210+w+5, 331, 429, filename);

  clear (209, 331, 430);

  if ( IS_RESTORE(0, 0, 0, filename) == 0 ) {
    get_filename (str, filename);
    IS_FRAME_COPY (0, 1);
  }
  else {
    error ();
  }

  print (320, 331, CENTER, BLACK, str);

  MouseShowPointer ();
}

void save_proc (void)
{
  int w;
  char str[50];

  MouseHidePointer ();

  clear (209, 331, 430);

  IS_SET_ACTIVE_REGION (0, 0, 512, 512);

  w = print (210, 331, LEFT, BLACK, "Filename:");

  input (210+w+5, 331, 429, filename);

  clear (209, 331, 430);

  if ( IS_SAVE (0, 0, 1, 0, filename) == 0 ) {
    get_filename (str, filename);
  }
  else {
    error ();
  }

  print (320, 331, CENTER, BLACK, str);

  MouseShowPointer ();
}

void flicker (void)
{
  int l;
  int r[512];

  for (l=0; l<512; l+=2) {
    IS_SET_ACTIVE_REGION (l, 0, 1, 512);
    IS_GET_REGION (l, r);
    IS_SET_ACTIVE_REGION (l+1, 0, 1, 512);
    IS_PUT_REGION (l, r);
  }
}

void acqu_proc (void)
{
  int norm = 1;
  int n, w;
  char str[50];

  wait_release();

  MouseHidePointer ();

  clear (209, 331, 430);

  IS_SET_SYNC_SOURCE (1);
  IS_PASSTHRU ();

  special_getch ();

  IS_FREEZE_FRAME ();
  IS_SET_SYNC_SOURCE(0);
  flicker ();
  IS_FRAME_COPY (1, 0);

  clear (209, 331, 430);

  w = print (210, 331, LEFT, BLACK, "Filename:");

  input (210+w+5, 331, 429, filename);

  clear (209, 331, 430);

  get_filename (str, filename);

  print (320, 331, CENTER, BLACK, str);
  MouseShowPointer ();
}
```

roin.c

```
#include <graph.h>

#include "mouse.h"
#include "spimut.h"

void roin_proc (void);
void draw_reg (void);

extern region SIZE;
extern region ROI;
extern region REG_A;
extern region REG_B;
extern region REG_C;
extern region REG_D;

void draw_cross (int x, int y)
{
  int a[2];

  IS_LOAD_MASK ( 14 );

  IS_SET_GRAPHIC_POSITION (y, 0);
  a[0] = y;
  a[1] = 511;
  IS_DRAW_LINES (1, 2, a);

  IS_SET_GRAPHIC_POSITION (0, x);
  a[0] = 511;
  a[1] = x;
  IS_DRAW_LINES (1, 2, a);

  IS_LOAD_MASK ( 0 );
}

void select (void)
{
  int static PointerCol_old = 127, PointerRow_old = 127;
  int ButtonStatus, PointerCol, PointerRow;
  int PointerCol_org, PointerRow_org;
  int x1, y1, x2, y2;
  char str[5];
  int n;

  MouseGetButtons(&ButtonStatus, &PointerCol_org,
    &PointerRow_org);

  MouseSetPointerHorizArea ( 0, 255);
  MouseSetPointerVertArea  ( 0, 255);

  MouseSetPointer ( PointerCol_old, PointerRow_old );

  clear (209, 331, 246);
  itoa (PointerCol_old*2, str, 10);
  print (228, 331, CENTER, BLACK, str);

  clear (393, 331, 430);
  itoa (PointerRow_old*2, str, 10);
  print (412, 331, CENTER, BLACK, str);

  IS_CURSOR(1);

  for (n= 0; n< 2; n++) {
    do {
      MouseGetButtons(&ButtonStatus, &PointerCol, &PointerRow);

      if ( PointerCol != PointerCol_old ) {
        IS_SET_CURSOR_POSITION( PointerRow*2, PointerCol*2 );
        PointerCol_old = PointerCol;
        clear (209, 331, 246);
        if (n == 0) itoa (PointerCol*2, str, 10);
        else        itoa (abs(x1-PointerCol*2), str, 10);
        print (228, 331, CENTER, BLACK, str);
      }

      if ( PointerRow != PointerRow_old ) {
        IS_SET_CURSOR_POSITION( PointerRow*2, PointerCol*2 );
        PointerRow_old = PointerRow;
        clear (393, 331, 430);
        if (n == 0) itoa (PointerRow*2, str, 10);
        else        itoa (abs(y1-PointerRow*2), str, 10);
        print (412, 331, CENTER, BLACK, str);
      }
    } while ( !ButtonStatus );

    wait_release();

    if ( n == 0 ) {
      x1 = PointerCol*2;
      y1 = PointerRow*2;
      IS_SET_FOREGROUND (255);
      draw_cross (x1, y1);
      clear (209, 331, 246);
      print (228, 331, CENTER, BLACK, "0");
      clear (393, 331, 430);
      print (412, 331, CENTER, BLACK, "0");
    }
    else {
      x2 = PointerCol*2;
      y2 = PointerRow*2;
      IS_SET_FOREGROUND (0);
      draw_cross (x1, y1);
    }
  }
  ROI.x  = (x1 < x2) ? x1 : x2;
  ROI.y  = (y1 < y2) ? y1 : y2;

  SIZE.x = abs (x2 - x1 + 1) / 5 * 5;
  SIZE.y = abs (y2 - y1 + 1) / 5 * 5;

  IS_CURSOR(0);

  clear (209, 331, 246);
  clear (393, 331, 430);

  MouseSetPointerHorizArea(0, 639);
  MouseSetPointerVertArea(0, 479);

  MouseSetPointer(PointerCol_org, PointerRow_org);
}

int too_big (void)
{
  if ( (SIZE.x > 250) || (SIZE.y > 250) ) return (TRUE);
  else return (FALSE);
}

int too_small (void)
{
  if ( (SIZE.x < 50) || (SIZE.y < 50) ) return (TRUE);
  else return (FALSE);
}

void draw_reg (void)
{
  int a[4][2];
  int x, y, w, h;

  x = ROI.x;
  y = ROI.y;
  w = SIZE.x;
  h = SIZE.y;

  IS_LOAD_MASK ( 14 );

  a[0][0] = y;
  a[0][1] = x + w - 1;
  a[1][0] = y + h - 1;
  a[1][1] = x + w - 1;
  a[2][0] = y + h - 1;
  a[2][1] = x;
  a[3][0] = y;
  a[3][1] = x;

  IS_SET_GRAPHIC_POSITION (y, x);
  IS_DRAW_LINES (1, 4, a);

  IS_LOAD_MASK ( 0 );
}

void change_regions (void)
{
  REG_A.x = REG_C.x = 252 - SIZE.x + 1;
  REG_B.x = REG_D.x = 259;

  REG_A.y = REG_B.y = 252 - SIZE.y + 1;
  REG_C.y = REG_D.y = 259;
}

void roin_proc (void)
{
  int done = 0;

  wait_release ();

  MouseHidePointer ();

  IS_DISPLAY (0);
  IS_FRAME_COPY (0, 1);
  IS_PERFORM_FEEDBACK (1, 5);
  IS_SELECT_OLUT (7);
  IS_DISPLAY (1);

  while ( !done ) {

    select ();

    if (!too_big() && !too_small())
      done = 1;
    else
      error();
  }

  IS_SET_FOREGROUND (255);
  draw_reg ();

  change_regions ();

  MouseShowPointer ();
}
```

proc.c

```
#include <graph.h>
#include <malloc.h>
#include <stdlib.h>
#include <time.h>

#include "mouse.h"
#include "spimut.h"

extern region SIZE;
extern region ROI;
extern region REG_A;
extern region REG_B;
extern region REG_C;
```

```
extern region REG_D;

extern int thrs;
extern int cmin;

extern results res;

IMAGE create_img (int ysize, int xsize);
void free_img (IMAGE img);
int retira_sup_base (IMAGE img, int k);
void equaliza (IMAGE img);
void averager (IMAGE in, IMAGE out, int k);
void localmin (IMAGE in, IMAGE cont, IMAGE dire);
void classifica (IMAGE thin, IMAGE org, IMAGE cont, IMAGE out);
void thin_img (IMAGE img);
void final_img (IMAGE thin, IMAGE gray, IMAGE out);
void calc_results (IMAGE thin, IMAGE final, IMAGE temp);
void display_results (void);
void save_img (IMAGE img, char *filename);
void clear_img (IMAGE img, int v);

int print (int x1, int y, int just, int color, char *str);
void special_getch (void);

#define PRETO    0
#define BRANCO 255

void buf2img (int x, int y, IMAGE img)
{
  int l, c;
  int buf [256];
  int *i;
  PIXEL *u;

  for (l=0; l<img.ysize; l++) {

    IS_SET_ACTIVE_REGION (y+l, x, 1, img.xsize);
    IS_GET_REGION (0, buf);

    i = buf;
    u = img.matrix[l];

    for (c=0; c<img.xsize; c++, u++, i++)
      *u = *i;
  }
}

void img2buf (IMAGE img, int x, int y, int bin)
{
  int l, c;
  int buf [256];
  int *i;
  PIXEL *u;

  for (l=0; l<img.ysize; l++) {

    i = buf;
    u = img.matrix[l];

    if (bin) {
      for (c=0; c<img.xsize; c++, u++, i++) {
        if (*u>128) *i = 128;
        else if (*u<77) *i = 179;
             else *i = 256-*u;
      }
    }
    else
      for (c=0; c<img.xsize; c++, u++, i++)
        *i = *u/2;

    IS_SET_ACTIVE_REGION (y+l, x, 1, img.xsize);
    IS_PUT_REGION (1, buf);
  }
}

void set_olut_6 (int v)
{
  int n;
  int lut[256];

  for (n=0; n<128; n++)
    lut[n] = n*2;

  for (n=128; n<129+v; n++)
    lut[n] = BRANCO;

  for (; n<256; n++)
    lut[n] = PRETO;

  IS_LOAD_OLUT (6, lut, lut, lut);
  IS_SELECT_CLUT (6);
}

void slice (int *p)
{
  int ButtonStatus;
  int PointerCol, PointerRow;
  int PointerCol_old;
  int PointerCol_org, PointerRow_org;
  char str[5];
  int w;
  int n;

  wait_release();

  MouseHidePointer ();

  set_olut_6 (*p);

  MouseGetButtons(&ButtonStatus, &PointerCol_org,
    &PointerRow_org);

  MouseSetPointerHorizArea(0, 500);

  PointerCol_old = *p;

  MouseSetPointer(PointerCol_old * 10, 0);

  clear (209, 331, 246);
  itoa (PointerCol_old, str, 10);
  print (228, 331, CENTER, BLACK, str);

  PointerCol_old = -1;

  do {

    MouseGetButtons(&ButtonStatus, &PointerCol, &PointerRow);
    PointerCol /= 10;

    if (PointerCol != PointerCol_old) {

      w = (PointerCol*108+25)/50;

      clear (265, 331, 374);

      _setcolor (LIGHTRED);
      _rectangle (_GFILLINTERIOR, 265, 331, 265+w, 343);

      _setcolor (RED);
      _moveto (265+w, 331);
      _lineto (265+w, 343);

      clear (393, 331, 430);
      itoa  (PointerCol, str, 10);
      print (412, 331, CENTER, BLACK, str);

      if (PointerCol > PointerCol_old)
        for (n=PointerCol_old; n<PointerCol; n++)
          IS_LOAD_OLUT_SVAL (6, 129+n, BRANCO, BRANCO, BRANCO);
      else
        for (n=PointerCol_old-1; n>=PointerCol; n--)
          IS_LOAD_OLUT_SVAL (6, 129+n, PRETO, PRETO, PRETO);

      PointerCol_old = PointerCol;
    }

  } while (!ButtonStatus);

  *p = PointerCol;

  wait_release ();

  clear (209, 331, 246);
  clear (265, 331, 374);
  clear (393, 331, 430);

  MouseSetPointerHorizArea(0, 639);

  MouseSetPointer(PointerCol_org, PointerRow_org);

  MouseShowPointer ();
}

void binacopy (IMAGE in, IMAGE out, int v)
{
  int l, c;
  PIXEL *i, *o;

  for (l=0; l<in.ysize; l++) {

    i = in.matrix[l];
    o = out.matrix[l];

    for (c=0; c<in.xsize; c++,i++,o++)
      *o = (*i>=128-v) ? 255 : 0;
  }

  margem (out, 3, 255);
}

void binadire (IMAGE in, IMAGE dir, int v)
{
  int l, c;
  PIXEL *i, *d;

  for (l=0; l<in.ysize; l++) {

    i = in.matrix[l];
    d = dir.matrix[l];

    for (c=0; c<in.xsize; c++,i++,d++)
      *i = (*i>=128-v) ? 255 : *d;
  }

  margem (in, 3, 255);
}

void message (char *str)
{
  _setcolor (LIGHTGRAY);
  _rectangle (_GFILLINTERIOR, 265, 331, 374, 343);

  _setfont( "t'helv' h10" );

  print (319, 332, CENTER, BLACK, str);

  _setfont( "t'helv' h12" );
}

#define PRE_AVERAGE 7
#define POS_AVERAGE 3

void process (int step_by_step)
{
```

```
IMAGE org, gray, cont, fim, thin, temp, esq, dir;

wait_release ();

MouseHidePointer ();

IS_FRAME_CLEAR (1);

set_olut_6 (0);

org  =        create_img (SIZE.y, SIZE.x);

buf2img (ROI.x, ROI.y, org);

img2buf (org, REG_A.x, REG_A.y, FALSE);

message ("Remocao sup. base");
retira_sup_base (org, 5);

gray =        create_img (SIZE.y, SIZE.x);
cont = fim  = create_img (SIZE.y, SIZE.x);
thin = temp = create_img (SIZE.y, SIZE.x);
esq  =        create_img (SIZE.y, SIZE.x);
dir  =        create_img (SIZE.y, SIZE.x);

message ("Equalizacao");
equaliza (org);
img2buf (org, REG_B.x, REG_B.y, FALSE);

message ("Deteccao min. locais");

averager (org, gray, PRE_AVERAGE);
localmin (gray, temp, dir);
averager (temp, cont, POS_AVERAGE);

if (step_by_step) {
  img2buf (cont, REG_C.x, REG_C.y, TRUE);
  slice (&thrs);
}
binacopy (cont, thin, thrs);

message ("Thinning");
thin_img (thin);

message ("Decomp. segmentos");
classifica (thin, gray, cont, esq);

message ("Classif. segmentos");
if (step_by_step) {
  img2buf (esq, REG_C.x, REG_C.y, TRUE);
  slice (&cmin);
}
binadire (esq, dir, cmin);
img2buf (esq, REG_C.x, REG_C.y, FALSE);

message ("Reconstrucao da rede");
final_img (esq, org, fim);
img2buf (fim, REG_D.x, REG_D.y, FALSE);

message ("Extr. caracteristicas");
calc_results (esq, fim, org);

free_img (org);
free_img (gray);
free_img (cont);
free_img (thin);
free_img (esq);
free_img (dir);

display_results ();

special_getch ();

MouseShowPointer ();
}

void auto_proc (void)
{
  process (FALSE);
}

void step_proc (void)
{
  process (TRUE);
}
```

para.c

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>

#include "spimut.h"

IMAGE create_img (int ysize, int xsize);
void free_img (IMAGE img);
void averager (IMAGE in, IMAGE out, int k);

//#define PARABOLOIDE

#ifdef PARABOLOIDE

int decima (IMAGE img, int k, int ***p_p, int *p_ny, int *p_nx)
{
  int **p;
  int ny, nx;
  int l, c;
  int y, x;
  int k2;
  long sum;
  PIXEL *q;
  int ll, cc;

  *p_ny = ny = img.ysize/k - 1;
  *p_nx = nx = img.xsize/k - 1;

  // aloca a matriz p

  if ((*p_p = p = (int **) malloc ((ny+2) * sizeof (int *)))
    == NULL) return (0);

  for (l=0; l<ny+2; l++)
    if ((p[l] = (int *) malloc ((nx+2) * sizeof (int)))
      == NULL) return (0);

  // preenche a matriz p com as medias

  k2 = k*k;
  y = k/2 + 1;
  x = k/2 + 1;

  for (l=0; l<ny; l++)
    for (c=0; c<nx; c++) {

      sum = 0;

      for (ll=0; ll<k; ll++) {
        q = &img.matrix[y+l*k+ll][x+c*k];

        for (cc=0; cc<k; cc++, q++)
          sum += *q;
      }

      p[l][c] = sum / k2;
    }

  return (1);
}

int constroi_sistema (int **p, int ny, int nx, double ***p_a,
  double **p_b)
{
  double **a;
  double *b;
  int l, c;
  double z;

  double x0, x1, x2, x3, x4;
  double y0, y1, y2, y3, y4;

  ny--, nx--;

  if ((*p_a = a = (double **) malloc (6 * sizeof(double *)))
    == NULL) return (0);

  for (l=0; l<6; l++)
    if ((a[l] = (double *) malloc (6 * sizeof(double)))
      == NULL) return (0);

  if ((*p_b = b = (double *) malloc (6 * sizeof(double)))
    == NULL) return (0);

  x0 = nx + 1;
  x1 = x0 * nx/2;
  x2 = x1 * (2*nx + 1)/3;
  x3 = x1 * x1;
  x4 = x2 * (3*nx*nx + 3*nx - 1)/5;

  y0 = ny + 1;
  y1 = y0 * ny/2;
  y2 = y1 * (2*ny + 1)/3;
  y3 = y1 * y1;
  y4 = y2 * (3*ny*ny + 3*ny - 1)/5;

  a[0][0] =                                   x4 * y0;
  a[0][1] = a[1][0] = a[4][4] =               x2 * y2;
  a[0][2] = a[2][0] =                         x3 * y0;
  a[0][3] = a[3][0] = a[2][4] = a[4][2] =     x2 * y1;
  a[0][4] = a[4][0] =                         x3 * y1;
  a[0][5] = a[5][0] = a[2][2] =               x2 * y0;
  a[1][1] =                                   x0 * y4;
  a[1][2] = a[2][1] = a[3][4] = a[4][3] =     x1 * y2;
  a[1][3] = a[3][1] =                         x0 * y3;
  a[1][4] = a[4][1] =                         x1 * y3;
  a[1][5] = a[5][1] = a[3][3] =               x0 * y2;
  a[2][3] = a[3][2] = a[4][5] = a[5][4] =     x1 * y1;
  a[2][5] = a[5][2] =                         x1 * y0;
  a[3][5] = a[5][3] =                         x0 * y1;
  a[5][5] =                                   x0 * y0;

  for (l=0; l<6; l++)
    b[l] = 0.0;

  for (l=0; l<=ny; l++)
    for (c=0; c<=nx; c++) {

      z = p[l][c];

      b[0] += c*c*z;
      b[1] += l*l*z;
      b[2] += c*z;
      b[3] += l*z;
      b[4] += c*l*z;
      b[5] += z;
    }

  return (1);
}
```

```
int resolve_sistema (double **p_x, int dim, double **a,
  double *b)
{
  double **m, **p;
  int n, l, c;
  double *x;
  int l_pivot, c_pivot;
  double temp;
  double *temp_p;

  if ((m = (double **) malloc (dim * sizeof(double *)))
    == NULL) return (0);

  for (l=0; l<dim; l++)
    if ((m[l] = (double *) malloc ((dim+1) * sizeof(double)))
      == NULL) return (0);

  for (l=0; l<dim; l++) {
    for (c=0; c<dim; c++)
      m[l][c] = a[l][c];
    m[l][dim] = b[l];
  }

  if ((p = (double **) malloc (dim * sizeof(double *)))
    == NULL) return (0);

  for (l=0; l<dim; l++)
    p[l] = &m[l][dim];

  for (n=0; n<dim; n++) {

    l_pivot = c_pivot = n;
    temp = 0.0;

    for (l=n; l<dim; l++)
      for (c=n; c<dim; c++)
        if (fabs(m[l][c]) > temp) {
          temp = fabs(m[l][c]);
          l_pivot = l;
          c_pivot = c;
        }

    if (!temp) return (0);

    if (l_pivot != n) {

      /* troca linha */

      temp_p = m[n];
      m[n] = m[l_pivot];
      m[l_pivot] = temp_p;

      temp_p = p[n];
      p[n] = p[l_pivot];
      p[l_pivot] = temp_p;
    }

    if (c_pivot != n) {

      /* troca coluna */

      for (l=0; l<dim; l++) {
        temp = m[l][n];
        m[l][n] = m[l][c_pivot];
        m[l][c_pivot] = temp;
      }

      temp_p = p[n];
      p[n] = p[c_pivot];
      p[c_pivot] = temp_p;
    }

    for (l=n+1; l<dim; l++)
      for (c=dim; c>n; c--)
        m[l][c] = m[l][c] - m[n][c] * m[l][n] / m[n][n];
  }

  for (l=dim-1; l>=0; l--) {
    for (c=l+1; c<dim; c++)
      m[l][dim] -= m[l][c] * m[c][dim];
    m[l][dim] /= m[l][l];
  }

  if ((*p_x = x = (double *) malloc (dim * sizeof(double)))
    == NULL) return (0);

  for (n=0; n<dim; n++)
    x[n] = *p[n];

  for (l=0; l<dim; l++)
    free (m[l]);
  free (m);
  free (p);

  for (n=0; n<6; n++)
    free (a[n]);
  free (a);

  free (b);

  return (1);
}

void constroi_paraboloide (int **p, int ny, int nx, double *x)
{
  int l, c;
  int *q;
  double t;
  long m;

  // calcula media

  m = 0;
  for (l=0; l<ny; l++)
    for (c=0; c<nx; c++)
      m += p[l][c];
  m = m / (ny*nx);

  ny++, nx++;

  // calcula paraboloide menos a media

  for (l=-1; l<ny; l++) {
    q = p[l+1];
    t = x[1]*l*l + x[3]*l + x[5] - m;
    for (c=-1; c<nx; c++)
      *q++ = (int) ( t + x[0]*c*c + x[2]*c + x[4]*c*l );
  }

  free (x);
}

int interpola (IMAGE img, int **p, int ny, int nx, int k)
{
  PIXEL *q;
  int l, c;
  int a, b, _a, _b;
  int *p00, *p01, *p10, *p11;
  int k2;
  int t;

  ny++, nx++;

  // interpola

  k2 = k*k;

  for (l=0; l<ny; l++) {
    for (a=0, _a=k; a<k; a++, _a--) {
      p00 = &p[l][0];
      p01 = p00 + 1;
      p10 = &p[l+1][0];
      p11 = p10 + 1;
      q = img.matrix[l*k+a];
      for (c=0; c<nx; c++) {
        for (b=0, _b=k; b<k; b++, _b--, q++) {

          t = *q - (_a*_b* *p00 + _a*b* *p01 + a*_b* *p10 + a*b*
            *p11) / k2;
          *q = (t > 255) ? 255 : t;
        }
        p00++, p01++, p10++, p11++;
      }
    }
  }

  // liberta a matriz p

  for (l=0; l<ny+1; l++)
    free (p[l]);

  free (p);

  return (1);
}

int retira_sup_base (IMAGE img, int k)
{
  int **p;
  double **a;
  double *b;
  double *x;
  int ny, nx;
  int n;

  if (!decima (img, k, &p, &ny, &nx)) return (0);

  if (!constroi_sistema (p, ny, nx, &a, &b)) return (0);

  if (!resolve_sistema (&x, 6, a, b)) return (0);

  constroi_paraboloide (p, ny, nx, x);

  if (!interpola (img, p, ny, nx, k)) return (0);

  return (1);
}

#else

void decima (IMAGE img, IMAGE *p_dec, int k)
{
  IMAGE dec;
  int ny, nx;
  int l, c;
  long sum;
  PIXEL *q;
  int ll, cc;

  ny = img.ysize/k+2;
  nx = img.xsize/k+2;

  *p_dec = dec = create_img (ny, nx);

  for (l=0; l<ny-2; l++)
    for (c=0; c<nx-2; c++) {

      sum = 0;

      for (ll=0; ll<k; ll++) {
        q = &img.matrix[l*k+ll][c*k];

        for (cc=0; cc<k; cc++, q++)
```

```
            sum += *q;
        }

        dec.matrix[l+1][c+1] = sum / (k*k);
      }


    for (c=1; c<nx-1; c++) {
      dec.matrix[   0][c] = dec.matrix[   1][c];
      dec.matrix[ny-1][c] = dec.matrix[ny-2][c];
    }

    for (l=0; l<ny; l++) {
      dec.matrix[l][   0] = dec.matrix[l][   1];
      dec.matrix[l][nx-1] = dec.matrix[l][nx-2];
    }
}

void filtra (IMAGE img, int k)
{
  IMAGE temp;

  temp = create_img (img.ysize, img.xsize);

  averager (img, temp, k);
  averager (temp, img, k);

  free_img (temp);
}

int interpola (IMAGE dec, IMAGE *p_sup, int k)
{
  IMAGE sup;
  PIXEL *q;
  int l, c;
  int a, b, _a, _b;
  PIXEL *p00, *p01, *p10, *p11;
  int k2;
  int t;
  long sum = 0;
  int m;

  for (l=0; l<dec.ysize; l++)
    for (c=0; c<dec.xsize; c++)
      sum += dec.matrix[l][c];

  m = sum / (dec.ysize * dec.xsize);

  *p_sup = sup = create_img ((dec.ysize-1)*k, (dec.xsize-1)*k);

  k2 = k*k;

  for (l=0; l<dec.ysize-1; l++) {
    for (a=0, _a=k; a<k; a++, _a--) {

      p00 = dec.matrix[l];
      p01 = p00 + 1;
      p10 = dec.matrix[l+1];
      p11 = p10 + 1;
      q   = sup.matrix[l*k+a];

      for (c=0; c<dec.xsize-1; c++) {

        for (b=0, _b=k; b<k; b++, _b--, q++)
          *q = (_a*_b* *p00 + _a*b* *p01 + a*_b* *p10 + a*b*
            *p11) / k2;

        p00++, p01++, p10++, p11++;
      }
    }
  }

  free_img (dec);

  return (m);
}

int retira_sup_base (IMAGE img, int k)
{
  IMAGE dec, sup;
  int med;
  PIXEL *i, *s;
  int l, c;
  int t;

  decima (img, &dec, k);

  filtra (dec, 5);

  med = interpola (dec, &sup, k);

  for (l=0; l<img.ysize; l++) {

    i = img.matrix[l];
    s = &sup.matrix[l+k/2][k/2];

    for (c=0; c<img.xsize; c++,i++,s++) {

      t = med + *i - *s;

      if (t < 0) *i = 0;
      else *i = (t > 255) ? 255 : t;
    }
  }

  free_img (sup);

  return (1);
}

#endif
```

oper.c

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <dos.h>

#include "spimut.h"

struct {
  int gray;
  int cont;
  int leng;
} segment [256];

void end_graph (void);

void copy_img (IMAGE org, IMAGE dest)
{
  int l;

  if ((org.xsize != dest.xsize) || (org.ysize != dest.ysize)) {
    end_graph ();
    printf("Diferent sizes!\n");
    printf("Press any key to halt:\n");
    getch ();
    exit( 1 );
  }

  for (l=0; l<org.ysize; l++)
    _fmemcpy (dest.matrix[l], org.matrix[l], org.xsize);
}

IMAGE create_img (int ysize, int xsize)
{
  IMAGE img;
  int l;

  if ((img.matrix = (MATRIX) malloc (ysize * sizeof (VECTOR)))
      == NULL) {
    end_graph ();
    printf("Alocation error!\n");
    printf("Press any key to halt:\n");
    getch ();
    exit( 1 );
  }

  for (l=0; l<ysize; l++)
    if ((img.matrix[l] = (VECTOR) malloc (xsize*sizeof (PIXEL)))
        == NULL) {
      end_graph ();
      printf("Alocation error!\n");
      printf("Press any key to halt:\n");
      getch ();
      exit( 1 );
    }

  img.ysize = ysize;
  img.xsize = xsize;

  return (img);
}

void free_img (IMAGE img)
{
  int l;

  for (l=0; l<img.ysize; l++)
    free (img.matrix[l]);

  free (img.matrix);
}

void save_img (IMAGE img, char *filename)
{
  struct {
    char id[2];
    long filesize;
    int reserved[2];
    long headersize;
    long infoSize;
    long width;
    long depth;
    int biPlanes;
    int bits;
    long biCompression;
    long biSizeImage;
    long biXPelsPerMeter;
    long biYPelsPerMeter;
    long biClrUsed;
    long biClrImportant;
  } bmp_head;

  struct {
    unsigned char b, g, r, x;
  } bmp_pal [256];

  FILE *file;
  int n;

  bmp_head.id[0]           = 'B';
  bmp_head.id[1]           = 'M';
  bmp_head.filesize        = (long) img.xsize*img.ysize + 1078;
  bmp_head.reserved[0]     = 0;
  bmp_head.reserved[1]     = 0;
```

```
bmp_head.headersize      = 1078;
bmp_head.infoSize        = 40;
bmp_head.width           = img.xsize;
bmp_head.depth           = img.ysize;
bmp_head.biPlanes        = 1;
bmp_head.bits            = 8;
bmp_head.biCompression   = 0;
bmp_head.biSizeImage     = (long) img.xsize * img.ysize;
bmp_head.biXPelsPerMeter = 4036;
bmp_head.biYPelsPerMeter = 4036;
bmp_head.biClrUsed       = 0;
bmp_head.biClrImportant  = 0;

for (n=0; n<256; n++) {

  bmp_pal[n].r = bmp_pal[n].g = bmp_pal[n].b = n;
  bmp_pal[n].x = 0;
}

file = fopen(filename, "wb");

fwrite (&bmp_head, sizeof(bmp_head), 1, file);

fwrite (&bmp_pal, sizeof(bmp_pal), 1, file);

for (n=img.ysize; n>0; n--)
  fwrite (img.matrix[n-1], 1, img.xsize, file);

fclose (file);
}

void margem (IMAGE img, int k, int v)
{
  int l, c;

  for (l=0; l<k; l++)
    for (c=0; c<img.xsize; c++)
      img.matrix[l][c] = img.matrix[img.ysize-l-1][c] = v;

  for (c=0; c<k; c++)
    for (l=0; l<img.ysize; l++)
      img.matrix[l][c] = img.matrix[l][img.xsize-c-1] = v;
}

#define IN_X(v)\
  (v>=0 && v<in.xsize)

#define IN_Y(v)\
  (v>=0 && v<in.ysize)

void averager (IMAGE in, IMAGE out, int k)
{
  int l, c;

  int sum;
  int cont;

  PIXEL *q;
  int ll, cc;
  int a, b;

  a = (k-1)/2;
  b = k-a;

  for (c=0; c<in.xsize; c++) {

    sum = 0;
    cont = 0;

    for (ll=-a; ll<b; ll++) {

      if (IN_Y(ll)) {

        for (cc=-a; cc<b; cc++)
          if (IN_X(c+cc)) {
            q = &in.matrix[ll][c+cc];
            break;
          }

        for ( ; cc<b; cc++)
          if (IN_X(c+cc)) {
            sum += *q++;
            cont++;
          }
      }
    }

    out.matrix[0][c] = sum / cont;

    for (l=1; l<in.ysize; l++) {

      if (IN_Y(l-a-1)) {

        for (cc=-a; cc<b; cc++)
          if (IN_X(c+cc)) {
            q = &in.matrix[l-a-1][c+cc];
            break;
          }

        for ( ; cc<b; cc++)
          if (IN_X(c+cc)) {
            sum -= *q++;
            cont--;
          }
      }

      if (IN_Y(l+b-1)) {

        for (cc=-a; cc<b; cc++)
          if (IN_X(c+cc)) {
            q = &in.matrix[l+b-1][c+cc];
            break;
          }
```

```
        }

        for ( ; cc<b; cc++)
          if (IN_X(c+cc)) {
            sum += *q++;
            cont++;
          }
      }

      out.matrix[l][c] = sum / cont;
    }
  }
}

void localmin (IMAGE in, IMAGE cont, IMAGE dire)
{
  int l, c;
  int d, nd;
  int dir;
  PIXEL *n_n, *n_ne, *n_e, *n_se, *n_s, *n_sw, *n_w, *n_nw;
  PIXEL *f_n, *f_ne, *f_e, *f_se, *f_s, *f_sw, *f_w, *f_nw;
  PIXEL *i, *pc, *pd;

  for (l=6; l<in.ysize-6; l++) {

    i    = &in.matrix[l][6];

    n_n  = &in.matrix[l-3][ 6];
    n_ne = &in.matrix[l-2][ 8];
    n_e  = &in.matrix[l  ][ 9];
    n_se = &in.matrix[l+2][ 8];
    n_s  = &in.matrix[l+3][ 6];
    n_sw = &in.matrix[l+2][ 4];
    n_w  = &in.matrix[l  ][ 3];
    n_nw = &in.matrix[l-2][ 4];

    f_n  = &in.matrix[l-6][ 6];
    f_ne = &in.matrix[l-4][10];
    f_e  = &in.matrix[l  ][12];
    f_se = &in.matrix[l+4][10];
    f_s  = &in.matrix[l+6][ 6];
    f_sw = &in.matrix[l+4][ 2];
    f_w  = &in.matrix[l  ][ 0];
    f_nw = &in.matrix[l-4][ 2];

    pc   = &cont.matrix[l][6];
    pd   = &dire.matrix[l][6];

    for (c=6; c<in.xsize-6; c++) {

      d = min (*n_n -*i, *n_s -*i);
      dir = 0;

      nd = min (*n_ne-*i, *n_sw-*i);
      if (nd > d) d = nd, dir = 1;

      nd = min (*n_e-*i, *n_w-*i);
      if (nd > d) d = nd, dir = 2;

      nd = min (*n_se-*i, *n_nw-*i);
      if (nd > d) d = nd, dir = 3;

      nd = min (*f_n-*i, *f_s-*i);
      if (nd > d) d = nd, dir = 0;

      nd = min (*f_ne-*i, *f_sw-*i);
      if (nd > d) d = nd, dir = 1;

      nd = min (*f_e-*i, *f_w-*i);
      if (nd > d) d = nd, dir = 2;

      nd = min (*f_se-*i, *f_nw-*i);
      if (nd > d) d = nd, dir = 3;

      if (d >  127) d =  127;
      if (d < -128) d = -128;

      *pc = 127-d;
      *pd = dir;

      i++;
      n_n++, n_ne++, n_e++, n_se++, n_s++, n_sw++, n_w++,n_nw++;
      f_n++, f_ne++, f_e++, f_se++, f_s++, f_sw++, f_w++,f_nw++;
      pc++;
      pd++;
    }
  }

  margem (cont, 6, 255);
}

void clear_img (IMAGE img, int v)
{
  int l, c;
  PIXEL *i, *o;

  for (l=0; l<img.ysize; l++) {

    i = &img.matrix[l][0];

    for (c=0; c<img.xsize; c++,i++)
      *i = v;
  }
}

void histogram (IMAGE img, long *h)
{
  int l, c;
  PIXEL *p;

  for (l=0; l<256; l++)
    h[l] = 0;
```

```
  for (l=0; l<img.ysize; l++) {
    p = img.matrix[l];
    for (c=0; c<img.xsize; c++, p++)
      h[*p]++;
  }
}

void aply_lut (IMAGE img, int *lut)
{
  int l, c;
  PIXEL *p;

  for (l=0; l<img.ysize; l++) {
    p = img.matrix[l];
    for (c=0; c<img.xsize; c++, p++)
      *p = lut[*p];
  }
}

void equaliza (IMAGE img)
{
  long hist[256];
  long sum, acum;
  int lut[256];
  int n;
  int inf, sup;

  histogram (img, hist);

  sum = 0;
  for (n=0; n<256; n++)
    sum += hist[n];

  acum = 0;
  for (n=0; n<256; n++) {
    acum += hist[n];
    if (acum > sum/100) break;
  }
  inf = n;

  acum = 0;
  for (n=255; n>=0; n--) {
    acum += hist[n];
    if (acum > sum/100) break;
  }
  sup = n;

  if (inf==sup) sup++;

  for (n=0; n<inf; n++)
    lut[n] = 0;

  for (n=inf; n<=sup; n++)
    lut[n] = (n-inf) * 255 / (sup-inf);

  for (n=255; n>sup; n--)
    lut[n] = 255;

  aply_lut (img, lut);
}

void erosion (IMAGE in, IMAGE out)
{
  int l, c;
  PIXEL *nw, *n, *ne, *w, *i, *e, *sw, *s, *se, *o;

  for (l=1; l<in.ysize-1; l++) {

    nw = out.matrix[l-1], n = nw+1, ne = n+1;
    w  = out.matrix[l  ], o = w+1,  e  = o+1;
    sw = out.matrix[l+1], s = sw+1, se = s+1;

    i = &in.matrix[l][1];

    for (c=1; c<in.xsize-1; c++) {

      if (*i==255)
        *nw = *n = *ne = *w = *o = *e = *sw = *s = *se = 255;
      else
        *o = 0;

      nw++, n++, ne++, w++, o++, e++, sw++, s++, se++, i++;
    }
  }
}

void dilatation (IMAGE in, IMAGE out)
{
  int l, c;
  PIXEL *nw, *n, *ne, *w, *i, *e, *sw, *s, *se, *o;

  for (l=1; l<in.ysize-1; l++) {

    nw = out.matrix[l-1], n = nw+1, ne = n+1;
    w  = out.matrix[l  ], o = w+1,  e  = o+1;
    sw = out.matrix[l+1], s = sw+1, se = s+1;

    i = &in.matrix[l][1];

    for (c=1; c<in.xsize-1; c++) {

      if (*i!=255)
        *nw = *n = *ne = *w = *o = *e = *sw = *s = *se = 0;
      else
        *o = 255;

      nw++, n++, ne++, w++, o++, e++, sw++, s++, se++, i++;
    }
  }
}
```

```
void classifica_pontos (IMAGE thin)
{
  PIXEL *p_border, *p, *n, *ne, *e, *se, *s, *sw, *w, *nw;
  int y, x;
  int c;

  for (y=1; y<thin.ysize-1; y++) {

    nw = thin.matrix[y-1]; n = nw+1; ne = n+1;
    w  = thin.matrix[y  ]; p = w+1;  e  = p+1;
    sw = thin.matrix[y+1]; s = sw+1; se = s+1;

    for (x=1; x<thin.xsize-1; x++) {

      if (*p!=255) {

        c = 1;

        if (*n !=255) c++;
        if (*ne!=255) c++;
        if (*e !=255) c++;
        if (*se!=255) c++;
        if (*s !=255) c++;
        if (*sw!=255) c++;
        if (*w !=255) c++;
        if (*nw!=255) c++;

        *p = (c==1) ? 255 : c-1;
      }

      p++, n++, ne++, e++, se++, s++, sw++, w++, nw++;
    }
  }
}

void retira_pontos (IMAGE img)
{
  PIXEL *p;
  int y, x;

  for (y=1; y<img.ysize-1; y++) {

    p = &img.matrix[y][1];

    for (x=1; x<img.xsize-1; x++,p++)

      if (*p!=1 && *p!=2) *p = 255;
  }
}

#define MINIMUM_LENGTH   5
#define SUFICIENT_LEVEL 60

void retira_arterias (IMAGE img, IMAGE out, int c)
{
  PIXEL *i, *o;
  int y, x;

  for (y=1; y<img.ysize-1; y++) {

    i = &img.matrix[y][1];
    o = &out.matrix[y][1];

    for (x=1; x<img.xsize-1; x++,i++,o++)

      if (*i>4 && *i<=c+4) {

        if (segment[*i-5].gray < SUFICIENT_LEVEL) *o = 0;
        else if (segment[*i-5].leng < MINIMUM_LENGTH) *o = 255;
             else *o = segment[*i-5].cont;
        *i = 255;
      }
  }
}

void track (IMAGE thin, IMAGE img, IMAGE cont, int c,
  int y, int x)
{
  int length, level, contrast;
  int v;
  int out = 0;
  int t;

  v = c+5;

  thin.matrix[y][x] = v;

  length = 1;
  level = img.matrix[y][x];
  contrast = cont.matrix[y][x];

  do {

    t = thin.matrix[--y][  x];
    if (t!=1 && t!=2) {

      t = thin.matrix[  y][++x];
      if (t!=1 && t!=2) {

        t = thin.matrix[++y][  x];
        if (t!=1 && t!=2) {

          t = thin.matrix[++y][  x];
          if (t!=1 && t!=2) {

            t = thin.matrix[  y][--x];
            if (t!=1 && t!=2) {

              t = thin.matrix[  y][--x];
              if (t!=1 && t!=2) {

                t = thin.matrix[--y][  x];
```

```
                if (t!=1 && t!=2)
                  t = thin.matrix[--y][  x];
              }
            }
          }
        }
      }

      if (t==1) out = 1;

      thin.matrix[y][x] = v;

      length++;
      level += img.matrix[y][x];
      contrast += cont.matrix[y][x];

    } while (!out);

    level /= length;
    contrast /= length;

    segment[c].gray = level;
    segment[c].cont = contrast;
    segment[c].leng = length;
  }

  void classifica (IMAGE thin, IMAGE org, IMAGE cont, IMAGE out)
  {
    PIXEL *i, *o;
    int y, x;
    int c = 0;

    clear_img (out, 255);

    classifica_pontos (thin);

    retira_pontos (thin);

    classifica_pontos (thin);

    for (y=1; y<thin.ysize-1; y++) {

      i = &thin.matrix[y][1];

      for (x=1; x<thin.xsize-1; x++,i++) {

        if (*i==1) track (thin, org, cont, c++, y, x);

        if (c==250) retira_arterias (thin, out, c), c = 0;
      }
    }

    if (c!=0) retira_arterias (thin, out, c);
  }
```

thin.c

```
#include "spimut.h"

#define NORTH 0
#define SOUTH 1
#define EAST  2
#define WEST  3

#define RETIRAVEL 2

int pode_retirar [256];

void preenche_array (void)
{
  int n;

  /* TODOS OS PONTOS PODEM SER RETIRADOS */

  for (n=0; n<256; n++)
    pode_retirar [n] = 1;

  /* EXCEPTUANDO OS PONTOS ISOLADOS */

  pode_retirar [  0] = 0;

  /* OS END POINTS */

  pode_retirar [  1] = 0;
  pode_retirar [  2] = 0;
  pode_retirar [  4] = 0;
  pode_retirar [  8] = 0;
  pode_retirar [ 16] = 0;
  pode_retirar [ 32] = 0;
  pode_retirar [ 64] = 0;
  pode_retirar [128] = 0;

  /* E OS 8 NAO-SIMPLE POINTS */

  for (n=0; n<256; n++) {

    /* .01 */
    /* . 0 */
    /* ... */
    if ( (n & 0x07) == 0x02 ) {
      pode_retirar [n] = 0;
      continue;
    }

    /* ... */
    /* . 0 */
```

```
    /* .01 */

    if ( (n & 0x1C) == 0x08 ) {
      pode_retirar [n] = 0;
      continue;
    }

    /* ... */
    /* 0 . */
    /* 10. */

    if ( (n & 0x70) == 0x20 ) {
      pode_retirar [n] = 0;
      continue;
    }

    /* 10. */
    /* 0 . */
    /* ... */

    if ( (n & 0xC1) == 0x80 ) {
      pode_retirar [n] = 0;
      continue;
    }

    /* .0. */
    /* 1 1 */
    /* .0. */

    if ( (n & 0x55) == 0x44 ) {
      pode_retirar [n] = 0;
      continue;
    }

    /* .1. */
    /* 0 0 */
    /* .1. */

    if ( (n & 0x55) == 0x11 ) {
      pode_retirar [n] = 0;
      continue;
    }
  }
}

int marca_pontos (IMAGE img, int border)
{
  PIXEL *p_border, *p, *n, *ne, *e, *se, *s, *sw, *w, *nw;
  int pontos_marcados;
  int y, x;
  int b;

  pontos_marcados = 0;

  for (y=1; y<img.ysize-1; y++) {

    nw = img.matrix[y-1]; n = nw+1; ne = n+1;
    w  = img.matrix[y  ]; p = w+1;  e  = p+1;
    sw = img.matrix[y+1]; s = sw+1; se = s+1;

    switch (border) {
      case NORTH: p_border = n;
            break;
      case SOUTH: p_border = s;
            break;
      case EAST : p_border = e;
            break;
      case WEST : p_border = w;
            break;
    }

    for (x=1; x<img.xsize-1; x++) {

      if (*p!=255 && *p_border==255) {

        b = 0;
        if (*n !=255) b |= 0x01;
        if (*ne!=255) b |= 0x02;
        if (*e !=255) b |= 0x04;
        if (*se!=255) b |= 0x08;
        if (*s !=255) b |= 0x10;
        if (*sw!=255) b |= 0x20;
        if (*w !=255) b |= 0x40;
        if (*nw!=255) b |= 0x80;

        if (pode_retirar[b]) {

          *p = RETIRAVEL;
          pontos_marcados++;
        }
      }

      p_border++;
      p++, n++, ne++, e++, se++, s++, sw++, w++, nw++;

    }
  }

  return (pontos_marcados);
}

void retira_pontos_marcados (IMAGE img)
{
  PIXEL *p;
  int y, x;
  for (y=1; y<img.ysize-1; y++) {

    p = &img.matrix[y][1];

    for (x=1; x<img.xsize-1; x++) {

      if (*p == RETIRAVEL) *p = 255;
```

```
      p++;
    }
  }
}

void thin_img (IMAGE img)
{
  int pontos_marcados;

  do {

    pontos_marcados = 0;

    pontos_marcados += marca_pontos (img, NORTH);

    retira_pontos_marcados (img);

    pontos_marcados += marca_pontos (img, SOUTH);

    retira_pontos_marcados (img);

    pontos_marcados += marca_pontos (img, EAST);

    retira_pontos_marcados (img);

    pontos_marcados += marca_pontos (img, WEST);

    retira_pontos_marcados (img);

  } while (pontos_marcados);
}
```

cara.c

```
#include <graph.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <stdio.h>
#include <math.h>

#include "mouse.h"
#include "spimut.h"

#define N_S   0
#define SW_NE 1
#define W_E   2
#define NW_SE 3

extern region SIZE;
extern region ROI;

extern char filename[50];

extern int thrs;
extern int cmin;

extern results res;

IMAGE create_img (int ysize, int xsize);
void free_img (IMAGE img);
void clear_img (IMAGE img, int v);
void margem (IMAGE img, int k, int v);
void erosion (IMAGE in, IMAGE out);
void dilatation (IMAGE in, IMAGE out);
void get_filename (char *out, char *in);

#define N 9

void extrai_perfil (int l, int c, IMAGE gray, int dir,
  PIXEL *perfil)
{
  int n;

  switch (dir) {

    case N_S:

      for (n=-N; n<=N; n++)
        *perfil++ = gray.matrix[l+n][c];

      break;

    case SW_NE:

      for (n=-N; n<=N; n++)
        *perfil++ = gray.matrix[l-n][c+n];

      break;

    case W_E:

      for (n=-N; n<=N; n++)
        *perfil++ = gray.matrix[l][c+n];

      break;

    case NW_SE:

      for (n=-N; n<=N; n++)
        *perfil++ = gray.matrix[l+n][c+n];

      break;
  }
}

int calcula_raio (PIXEL *perfil, int dir)
{
  int n;
  PIXEL *p;
  int min, max, med;
  int raio, raio1, raio2;

  min =255;
  max = 0;
  p = perfil;

  for (n=0; n<2*N+1; n++,p++) {
    if (*p<min) min = *p;
    if (*p>max) max = *p;
  }

  med = (min + max) / 2;

  raio1 = 1;
  n = N+1;
  while (perfil[n] < med && n<2*N+1) {
    n++;
    raio1++;
  }

  raio2 = 1;
  n = N-1;
  while (perfil[n] < med && n>=0) {
    n--;
    raio2++;
  }

  raio = min (raio1, raio2);

  if (dir==1 || dir==3) raio = (raio*14+5)/10;

  return (raio/2);
}

#define LX      2
#define LY      2
#define LY_2 LY/2

void desenha_bola (int y, int x, int raio, IMAGE img, int v)
{
  int l, c;
  PIXEL *p;
  int a, b;
  int a2, b2;
  int a2b2;
  int t;

  a = raio;
  b = (raio * LX + LY_2) / LY;

  a2 = a*a;
  b2 = b*b;

  a2b2 = a2*b2;

  for (l=-b; l<=b; l++) {

    p = &img.matrix[y+l][x-a];
    t = a2*l*l;

    for (c=-a; c<=a; c++,p++)
      if ( b2*c*c+t <= a2b2 ) *p = v;
  }
}

void final_img (IMAGE thin, IMAGE gray, IMAGE out)
{
  int l, c;
  PIXEL *p;
  PIXEL perfil[2*N+1];

  clear_img (out, 255);

  for (l=N; l<thin.ysize-N; l++) {

    p = &thin.matrix[l][N];

    for (c=N; c<thin.xsize-N; c++,p++)

      if (*p != 255) {

        extrai_perfil (l, c, gray, *p, perfil);

        *p = calcula_raio (perfil, *p);

        desenha_bola (l, c, min(*p, 6), out, 0);

      }
  }

  dilatation (out, gray);

  erosion (gray, out);
}

void change_end_points (IMAGE thin)
{
  PIXEL *p_border, *p, *n, *ne, *e, *se, *s, *sw, *w, *nw;
  int y, x;
  int c;

  for (y=1; y<thin.ysize-1; y++) {

    nw = thin.matrix[y-1]; n = nw+1; ne = n+1;
    w  = thin.matrix[y  ]; p = w+1;  e  = p+1;
    sw = thin.matrix[y+1]; s = sw+1; se = s+1;

    for (x=1; x<thin.xsize-1; x++) {
```

```
      if (*p!=255) {

        c = 1;

        if (*n !=255) c++;
        if (*ne!=255) c++;
        if (*e !=255) c++;
        if (*se!=255) c++;
        if (*s !=255) c++;
        if (*sw!=255) c++;
        if (*w !=255) c++;
        if (*nw!=255) c++;

        if (c==2) *p += 128;
      }

      p++, n++, ne++, e++, se++, s++, sw++, w++, nw++;
    }
  }
}

void track_thin (IMAGE thin, int y, int x, int *length,
  int *cmed)
{
  int out = 0;
  int t;
  int len;
  int sum;

  len = 1;
  sum = thin.matrix[y][x] - 128;

  thin.matrix[y][x] = 255;

  do {

    t = thin.matrix[--y][  x];
    if (t==255) {

      t = thin.matrix[  y][++x];
      if (t==255) {

        t = thin.matrix[++y][  x];
        if (t==255) {

          t = thin.matrix[++y][  x];
          if (t==255) {

            t = thin.matrix[  y][--x];
            if (t==255) {

              t = thin.matrix[  y][--x];
              if (t==255) {

                t = thin.matrix[--y][  x];
                if (t==255)

                  t = thin.matrix[--y][  x];
              }
            }
          }
        }
      }
    }

    if (t>=128) out = 1;

    thin.matrix[y][x] = 255;

    len++;

    if (t >= 128)
      sum += (t-128);
    else
      sum += t;

  } while (!out);

  *length = len;
  *cmed = (sum + len/2) / len * 2 + 1;
}

void topologia (IMAGE thin)
{
  int l, c;
  PIXEL *p;
  int length, calibr;
  int len_sum, sum;
  int nseg;

  nseg = 0;

  res.lmin = 1000;
  res.lmax = 0;

  res.cmin = 1000;
  res.cmax = 0;

  len_sum = 0;
  sum = 0;

  for (l=1; l<thin.ysize-1; l++) {

    p = &thin.matrix[l][1];

    for (c=1; c<thin.xsize-1; c++,p++)

      if (*p!=255 && *p>=128) {

        track_thin (thin, l, c, &length, &calibr);

        nseg++;
```

```
        len_sum += length;

        if (length < res.lmin) res.lmin = length;
        if (length > res.lmax) res.lmax = length;

        sum += length * calibr;

        if (calibr < res.cmin) res.cmin = calibr;
        if (calibr > res.cmax) res.cmax = calibr;
      }
  }

  if (!nseg) {
    res.lmin = 0;
    res.lmed = 0;
    res.cmin = 0;
    res.cmed = 0;
  }
  else {
    res.lmed = (len_sum + nseg/2) / nseg;
    res.cmed = (sum + len_sum/2) / len_sum;
  }

  res.nseg = 100.0 * nseg / ((float) res.w * res.h);
}

#define MARGEM 5

#define QUAD 10

#define AREA 175
#define PERI  75
#define NODO   0

void img_area_peri (IMAGE in, IMAGE out)
{
  int l, c;
  PIXEL *n, *w, *i, *e, *s, *o;

  for (l=MARGEM; l<in.ysize-MARGEM; l++) {

    n = &in.matrix[l-1][MARGEM];                    //   N
    i = &in.matrix[l  ][MARGEM], w = i-1, e = i+1;  // W I E
    s = &in.matrix[l+1][MARGEM];                    //   S

    o = &out.matrix[l ][MARGEM];

    for (c=MARGEM;c<in.xsize-MARGEM;c++,n++,w++,e++,s++,i++,o++)

      if (!*i) {
        *o = AREA;
        if ((*i*4 - *n - *w - *e - *s) < 0)
          *o = PERI;
      }
      else
        *o = 255;
  }
}

#define RAIO 4

void img_nodos (IMAGE in, IMAGE out)
{
  int l, c;
  PIXEL *i, *o;

  change_end_points (in);

  for (l=0; l<in.ysize; l++) {

    i =  in.matrix [l];
    o = out.matrix [l];

    for (c=0; c<in.xsize; c++,i++,o++)
      if (*i!=255)
        *o = (*i>=128) ? NODO : 255;
  }

  for (l=MARGEM; l<out.ysize-MARGEM; l++) {

    o = &out.matrix [l][MARGEM];

    for (c=MARGEM; c<out.xsize-MARGEM; c++,o++)
      if (!*o) {
        desenha_bola (l, c, RAIO, out, 255);
        *o = NODO;
      }
  }
}

float set_grid (IMAGE img, int v, IMAGE grid)
{
  int l, c;
  long acum = 0L;
  PIXEL *i;

  clear_img (grid, 0);

  for (l=MARGEM; l<img.ysize-MARGEM; l++) {

    i = &img.matrix [l][MARGEM];

    for (c=MARGEM; c<img.xsize-MARGEM; c++)
      if (*i++ <= v) {
        grid.matrix [(l-MARGEM)/QUAD][(c-MARGEM)/QUAD]++;
        acum++;
      }
  }

  return (100.0 * acum / ((float) res.w * res.h));
}
```

```
void fill_grid (IMAGE img, IMAGE grid)
{
  int l, c;
  IMAGE acum;
  PIXEL *i;

  acum = create_img (grid.ysize-1, grid.xsize-1);

  clear_img (acum, 0);

  clear_img (grid, 0);

  for (l=MARGEM; l<img.ysize-MARGEM; l++) {

    i = &img.matrix [l][MARGEM];

    for (c=MARGEM; c<img.xsize-MARGEM; c++,i++)
      if (*i != 255) {
        grid.matrix [(l-MARGEM)/QUAD][(c-MARGEM)/QUAD] += *i;
        acum.matrix [(l-MARGEM)/QUAD][(c-MARGEM)/QUAD]++;
      }
  }

  for (l=0; l<acum.ysize; l++)
    for (c=0; c<acum.xsize; c++)
      if ( acum.matrix [l][c] )
        grid.matrix[l][c] = (grid.matrix[l][c] +
          acum.matrix[l][c]/2) / acum.matrix[l][c];
      else
        grid.matrix[l][c] = 0;

  free_img (acum);
}

float homo (IMAGE grid)
{
  int y, x;
  int ny, nx;
  int sum;
  float my, mx;
  float xx, yy;
  int t;

  ny = grid.ysize-1;
  nx = grid.xsize-1;

  sum = 0;
  my = mx = 0.0;

  for (y=0; y<ny; y++)
    for (x=0; x<nx; x++) {

      t = grid.matrix [y][x];

      sum += t;

      my += (float) t * y;
      mx += (float) t * x;
    }

  my /= sum;
  mx /= sum;

  yy = xx = 0.0;

  for (y=0; y<ny; y++)
    for (x=0; x<nx; x++) {

      yy += (my - y) * (my - y) * grid.matrix [y][x];
      xx += (mx - x) * (mx - x) * grid.matrix [y][x];
    }

  yy /= (sum * ny);
  xx /= (sum * nx);

  return ( pow (yy*xx, 1.0/6) );
}

void calc_results (IMAGE thin, IMAGE final, IMAGE temp)
{
  struct _dostime_t t;
  struct _dosdate_t d;
  IMAGE grid;
  char str[50];

  _dos_gettime (&t);
  sprintf (res.time, "%02d:%02d:%02d",t.hour,t.minute,t.second);

  _dos_getdate (&d);
  sprintf (res.date, "%02d-%02d-%02d",d.day,d.month,d.year%100);

  get_filename (str, filename);
  strcpy (res.name, str);

  res.x    = ROI.x  +     MARGEM;
  res.y    = ROI.y  +     MARGEM;
  res.w    = SIZE.x - 2 * MARGEM;
  res.h    = SIZE.y - 2 * MARGEM;

  res.thrs = thrs;
  res.cont = cmin;
  grid = create_img (res.h/QUAD+1, res.w/QUAD+1);

  img_area_peri (final, temp);

  res.area = set_grid (temp, AREA, grid);
  res.hare = homo (grid);

  res.peri = set_grid (temp, PERI, grid);
  res.hper = homo (grid);

  fill_grid (temp, grid);
  res.hdia = homo (grid);

  img_nodos (thin, temp);

  res.nnod = set_grid (temp, NODO, grid);
  res.hnod = homo (grid);

  free_img (grid);

  topologia (thin);
}
```

resl.c

```
#include <graph.h>
#include <stdlib.h>
#include <dos.h>
#include <string.h>
#include <stdio.h>

#include "mouse.h"
#include "spimut.h"

extern region SIZE;
extern region ROI;

extern char filename[50];

extern char fic_name[50];

extern int thrs;
extern int cmin;

extern results res;

int print (int x1, int y, int just, int color, char *str);
void error (void);
void get_filename (char *out, char *in);
void input (int x1, int y, int x2, char *str);

#define XL 192
#define XR 447
#define YY 261

#define NC  21

void draw_paper (void)
{
  int n;
  int y = YY;

  _setcolor (BLACK);
  _rectangle (_GFILLINTERIOR, XL, 260, XR, 359);

  _setcolor (WHITE);
  _rectangle (_GFILLINTERIOR, XL, y, XL+18, y+NC*10-1);
  _rectangle (_GFILLINTERIOR, XR-18, y, XR, y+NC*10-1);

  _setcolor (WHITE);
  for (n=XL+2; n<=XR; n+=10)
    _setpixel (n, y-1);

  for (n=0; n<NC; n++) {

    if (n%3 == 1) {
      _setcolor (BLACK);
      _ellipse (_GFILLINTERIOR, XL+5, y, XL+13, y+8);
      _ellipse (_GFILLINTERIOR, XR-13, y, XR-5, y+8);
    }

    if ((n/3) % 2) {
      _setcolor (WHITE);
      _rectangle (_GFILLINTERIOR, XL+19, y, XR-19, y+8);
      y += 9;
      _setcolor (LIGHTGRAY);
      _moveto (XL+19, y);
      _lineto (XR-19, y);
      y++;
    }
    else {
      _setcolor (LIGHTGRAY);
      _rectangle (_GFILLINTERIOR, XL+19, y, XR-19, y+8);
      y += 9;
      _setcolor (WHITE);
      _moveto (XL+19, y);
      _lineto (XR-19, y);
      y++;
    }
  }

  _setcolor (WHITE);
  for (n=XL+2; n<=XR; n+=10)
    _setpixel (n, y);
}

void print_par (char *s, int l, int y, int v)
{
  char str[20];

  print (l, y, CENTER, BLACK, s);
  sprintf (str, "%d", v);
  print (l, y+10, CENTER, RED, str);
}

#define PERCT 0
#define INTEG 1
#define FLOAT 2
```

```
void print_res (int *y, char *desgn, void *p_v, int type)
{
  char str[20];
  char units[3];

  *y += 10;

  print (XL+25, *y, LEFT, BLACK, desgn);

  switch (type) {

    case PERCT: {
      sprintf (str, "%8.5f", *((float *) p_v));
      strcpy (units, "%");
      break;
    }

    case INTEG: {
      sprintf (str, "%d", *((int *) p_v));
      strcpy (units, "p");
      break;
    }

    case FLOAT: {
      sprintf (str, "%.5f", *((float *) p_v));
      strcpy (units, "");
      break;
    }
  }

  print (XR-37, *y, RIGHT, RED, str);
  print (XR-32, *y, LEFT, BLACK, units);
}

void display_results (void)
{
  int y = YY;
  char str[20];

  draw_paper ();

  _setfont( "t'helv' h10" );

  print (320, y=YY-1, CENTER, RED, res.name);

  y += 20;

  print_par ("X", 320-90, y, res.x);
  print_par ("Y", 320-60, y, res.y);
  print_par ("W", 320-30, y, res.w);
  print_par ("H", 320   , y, res.h);

  print_par ("MPC", 320+60, y, res.thrs);
  print_par ("MSC", 320+90, y, res.cont);

  y += 20;

  print_res (&y, "Area vascular",
    (void *) &res.area, PERCT);
  print_res (&y, "Perimetro vascular",
    (void *) &res.peri, PERCT);
  print_res (&y, "Numero de segmentos",
    (void *) &res.nseg, PERCT);
  print_res (&y, "Numero de nodos",
    (void *) &res.nnod, PERCT);
  print_res (&y, "Calibre minimo",
    (void *) &res.cmin, INTEG);
  print_res (&y, "Calibre medio",
    (void *) &res.cmed, INTEG);
  print_res (&y, "Calibre maximo",
    (void *) &res.cmax, INTEG);
  print_res (&y, "Comprimento minimo",
    (void *) &res.lmin, INTEG);
  print_res (&y, "Comprimento medio",
    (void *) &res.lmed, INTEG);
  print_res (&y, "Comprimento maximo",
    (void *) &res.lmax, INTEG);
  print_res (&y, "Homogeneidade de area",
    (void *) &res.hare, FLOAT);
  print_res (&y, "Homogeneidade de perimetro",
    (void *) &res.hper, FLOAT);
  print_res (&y, "Homogeneidade de calibre",
    (void *) &res.hdia, FLOAT);
  print_res (&y, "Homogeneidade de nodos",
    (void *) &res.hnod, FLOAT);

  print (320-50, y+=20, CENTER, RED, res.time);
  print (320+50, y    , CENTER, RED, res.date);

  _setfont( "t'helv' h12" );
}

int exist_fic (void)
{
  FILE *fic;

  fic = fopen (fic_name, "r");

  if ( fic == NULL ) return (FALSE);

  fclose (fic);

  return (TRUE);
}

int create_fic (void)
{
  FILE *fic;

  if ((fic = fopen (fic_name, "w+")) == NULL)
    return (FALSE);

  fprintf (fic, "\
    FILENAME\t\
    AREA\t\
    PERI\t\
    NSEG\t\
    NNOD\t\
CMIN\t\
CMED\t\
CMAX\t\
LMIN\t\
LMED\t\
LMAX\t\
    HARE\t\
    HPER\t\
    HCAL\t\
    HNOD\t\
  X\t\
  Y\t\
  W\t\
  H\t\
MPC\t\
MSC\t\
    TIME\t\
    DATE\n");

  fclose (fic);

  return (TRUE);
}

void send2file (void)
{
  FILE *fic;

  if ( (fic = fopen (fic_name, "a+")) == NULL) return;

  fprintf (fic, "\
%12s\t\
%8.5f\t\
%8.5f\t\
%8.5f\t\
%8.5f\t\
%4d\t\
%4d\t\
%4d\t\
%4d\t\
%4d\t\
%4d\t\
%8.6f\t\
%8.6f\t\
%8.6f\t\
%8.6f\t\
%3d\t\
%3d\t\
%3d\t\
%3d\t\
%3d\t\
%3d\t\
%8s\t\
%8s\n",

    res.name,

    res.area,
    res.peri,
    res.nseg,
    res.nnod,
    res.cmin,
    res.cmed,
    res.cmax,
    res.lmin,
    res.lmed,
    res.lmax,
    res.hare,
    res.hper,
    res.hdia,
    res.hnod,

    res.x,
    res.y,
    res.w,
    res.h,

    res.thrs,
    res.cont,

    res.time,
    res.date);

  fclose (fic);
}

void disk_proc (void)
{
  if (!exist_fic ())
    if (!create_fic ())
      return;

  send2file ();
}

void name_proc (void)
{
  int w;  char str [15];

  clear (209, 331, 430);

  w = print (210, 331, LEFT, BLACK, "Filename:");

  input (210+w+5, 331, 429, fic_name);
```

```
  clear (209, 331, 430);

  get_filename (str, fic_name);
  print (320, 331, CENTER, BLACK, str);
}

int send2printer (char *str)
{
  fprintf (stdprn, "%s", str);

  if ( ferror (stdprn) ) return (FALSE);

  fprintf (stdprn, "%c%c", 13, 10);

  if ( ferror (stdprn) ) return (FALSE);

  return (TRUE);
}

#define ERR if ( !send2printer (str) ) return

void prnt_proc (void)
{
  char str[60];

  if ( !send2printer ("") ) return;

  if ( !send2printer
("************************************************") )
    return;
  if ( !send2printer ("*
*") )
    return;

  sprintf (str, "*  %-12s%20s    %s  *", res.name, res.time,
res.date); ERR;

  if ( !send2printer ("*
*") )
    return;

  sprintf (str, "*%18c%13d%18c", 'X', res.x, '*'); ERR;
  sprintf (str, "*%18c%13d%18c", 'Y', res.y, '*'); ERR;
  sprintf (str, "*%18c%13d%18c", 'W', res.w, '*'); ERR;
  sprintf (str, "*%18c%13d%18c", 'H', res.h, '*'); ERR;

  if ( !send2printer ("*
*") )
    return;

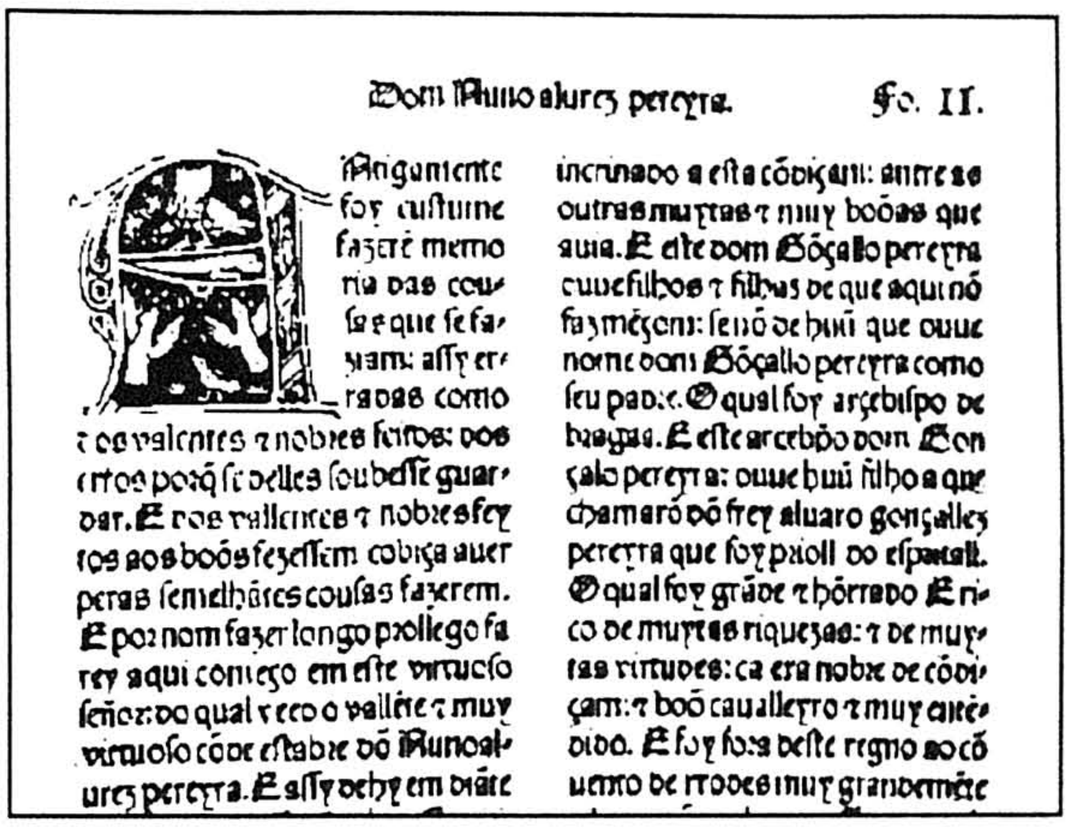
  sprintf (str, "*%19s%12d%18c", "MPC", res.thrs, '*'); ERR;
  sprintf (str, "*%19s%12d%18c", "MSC", res.cont, '*'); ERR;

  if ( !send2printer ("*
*") )
    return;

  sprintf (str, "*  Area vascular                %8.5f **
*", res.area); ERR;
  sprintf (str, "*  Perimetro vascular           %8.5f **
*", res.peri); ERR;
  sprintf (str, "*  Numero de segmentos          %8.5f **
*", res.nseg); ERR;
  sprintf (str, "*  Numero de nodos              %8.5f **
*", res.nnod); ERR;
  sprintf (str, "*  Calibre minimo               %8d p  *",
res.cmin); ERR;
  sprintf (str, "*  Calibre medio                %8d p  *",
res.cmed); ERR;
  sprintf (str, "*  Calibre maximo               %8d p  *",
res.cmax); ERR;
  sprintf (str, "*  Comprimento minimo           %8d p  *",
res.lmin); ERR;
  sprintf (str, "*  Comprimento medio            %8d p  *",
res.lmed); ERR;
  sprintf (str, "*  Comprimento maximo           %8d p  *",
res.lmax); ERR;
  sprintf (str, "*  Homogeneidade de area        %8.5f
*",  res.hare); ERR;
  sprintf (str, "*  Homogeneidade de perimetro   %8.5f
*",  res.hper); ERR;
  sprintf (str, "*  Homogeneidade de calibre     %8.5f
*",  res.hdia); ERR;
  sprintf (str, "*  Homogeneidade de nodos       %8.5f
*",  res.hnod); ERR;

  if ( !send2printer ("*
*") )
    return;
  if ( !send2printer
("************************************************") )
    return;

  if ( !send2printer ("") ) return;
}
```



APÊNDICE C

ARTIGO

Neste apêndice é incluído um artigo escrito sobre este trabalho e que foi submetido à 3rd International Conference on Computer Graphics and Image Processing (GKPO'94).

A Computer Based System of Uterine Hysteroscopic Images Analysis

Hélio Mendonça [1], **J. P. Marques de Sá** [1], **J. L. Silva Carvalho** [2]

[1] Faculdade de Engenharia da Universidade do Porto, DEEC, Porto, Portugal
[2] Hospital de S. João, Serviço de Ginecologia, Porto, Portugal

Abstract - **An objective determination of uterine vascular network features is of great interest in the study of abnormal situations like cancer and infertility. This is however an almost impossible task to achieve by plain visual inspection. Also the automatic analysis of these hysteroscopic images is very difficult due to their characteristics. This fact forced the development of an alternative method of vascular network detection. This method has simultaneously a local and global approach which makes it fast and suitable to producing good results even when the vascular network is very dimmed and has a weak contrast.**

Index Terms - **Vascular network detection, Base surface removal, Local minimum detection, Thinning, Image Processing, Medical imaging, Uterine hysteroscophy.**

I. INTRODUCTION

The blood irrigation performed by vascular networks seems to be strongly related with the regular functioning of the tissues irrigated by them. The study of the eventual correlation between vascular network features and certain physiological and pathological conditions like infertility and cancer is presently carried out at S. João Hospital, Porto. Networks with small area and composed by small calibre vessels seem to reveal infertility. On the other hand high vascular areas and thick vessels are unequivocal signs of cancer.

In order to obtain a set of features from the vascular network in an objective and quantitative way we developed a computer based system of uterine hysteroscopic images analysis (SPIMUT). This system is capable of process images previously recorded on video tapes and provide the extracted features for subsequent statistical analysis.

The uterine images are obtained by an hysteroscopic process. As can be seen in Fig. 1, a probe attached on a camera is inserted into the uterus distended by carbon dioxide gas. The images to be processed refer to the uterus fundus where the uterine mucosal (endometrium) with its vascular network come into view. These images are acquired a few millimetres away from the uterine wall.

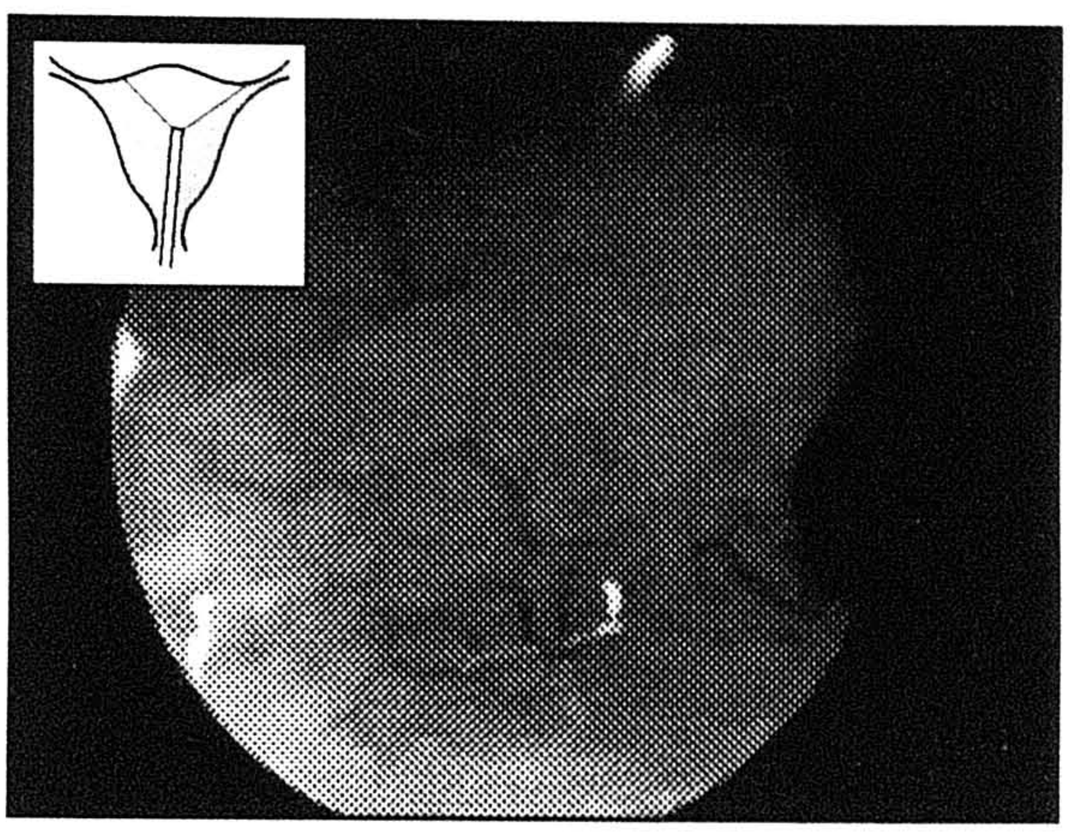


Fig. 1. Hysteroscopic image of the uterus

Gathering of this kind of images causes a lot of discomfort which makes this process very brief and consequently incautious in terms of focalization and illumination. The images may also present some artifacts like reflections, blood spots and bubbles caused by gas. On certain stages of the menstrual cycle, vessels are situated inside of the endometrium which makes them dimmed and with low contrast. These facts turn the vascular network detection (crucial stage of all processing) a difficult task incapable of being properly accomplished by other methods like Thresholding [1], Edge Detection [2, 3] and Vessel Tracking [4, 5]. The development of an alternative method is consequently essential.

II. PRE-PROCESSING

Pre-processing tries to enhance the contrast of the image in order to assist the subsequent stage (vascular network detection). This stage is performed by two operations: base surface removal and histogram equalization. The first one removes the intensity variations of the image background which also reduces the intensity range allowing a large image contrast enhancement by equalization. Next, both operations are fully explained.

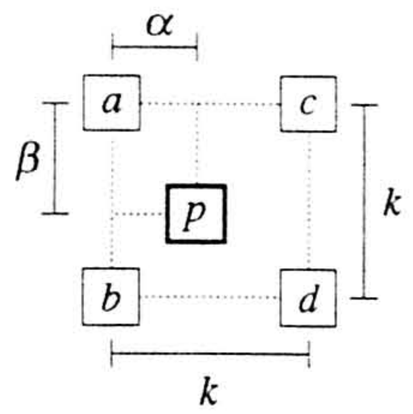
A. Base Surface Removal

Through this operation the original image will be subtracted by the base surface. To avoid negative values the surface average intensity is added.

$$out = in - surface + \mu_{surface} \quad (1)$$

Base surface is obtained using a similar method to the one already applied with success on impedance cardiographic signal processing [6].

The image is decimated by a k factor. In the new image (with less number of pixels) each pixel represents the average intensity of a k x k window in the original image. Then the surface is estimated and finally a bilinear interpolation using the same k factor is performed.



$$p = \frac{(k-\alpha).(k-\beta).a + (k-\alpha).\beta.b + \alpha.(k-\beta).c + \alpha.\beta.d}{k^2} \qquad (2)$$

The developed system estimates base surface through two consecutive filtering operations with an average filter (dimension 5). The k factor used in decimation and interpolation has also a value of 5.

B. Histogram Equalization

There are two basic processes to perform an histogram equalization: a linear and a non-linear way.

In a non-linear equalization method, histogram stripes are separated proportionally to their amplitude. That is, stripes representing intensities with large number of pixels will be further separated than stripes with lower weight. This fact in certain situations can lead to a higher contrast enhancement but in many others a loss of information may occur since certain important but small stripes will be melted into a unique intensity.

Therefore a linear equalization was adopted. With this mechanism, once low and high histogram limits are defined, they are respectively pulled to black and white intensity levels. The stripes are equally spaced no matter their amplitude.

In the following figures it is possible to see the results of these two operations. In spite of the acquired images having a 512 x 512 dimension only a restricted region of interest (ROI) defined by the user is processed based on its clinical significance.

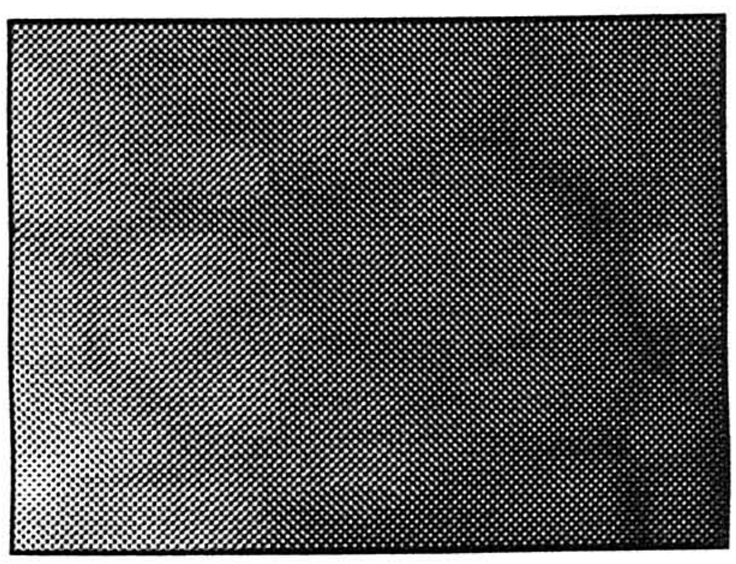


Fig. 2. Original ROI

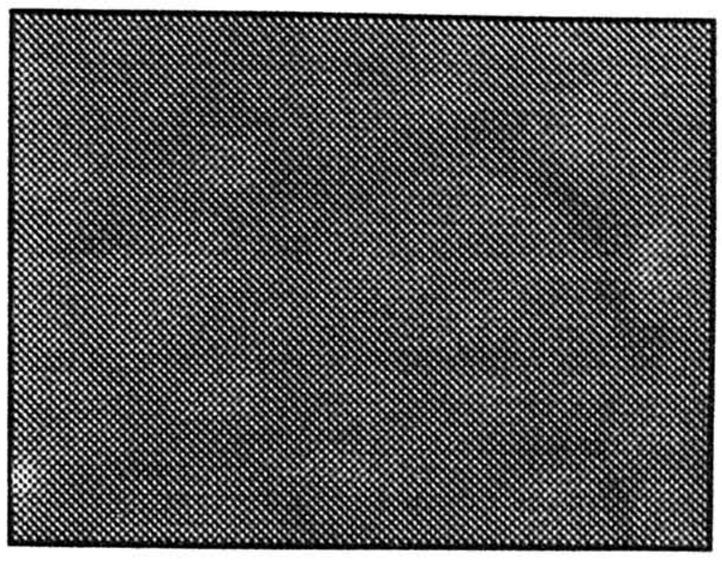


Fig. 3. ROI with base surface removed

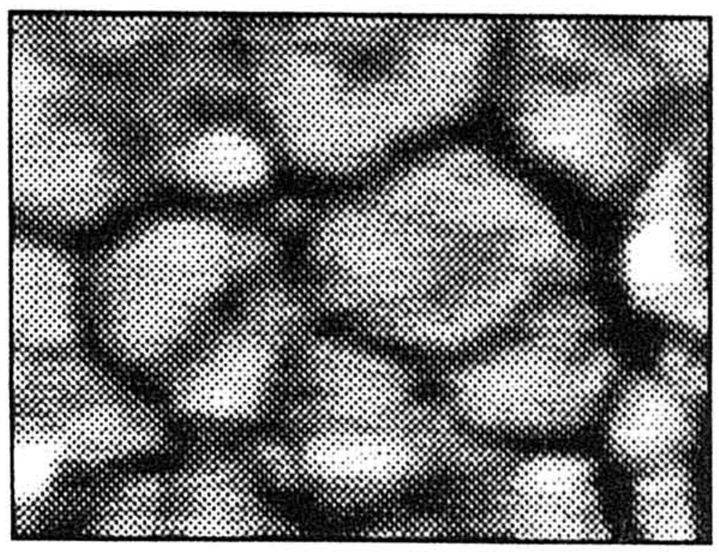


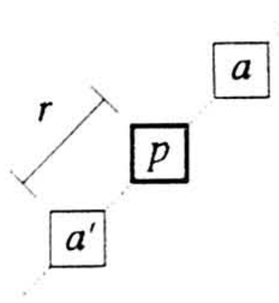
Fig. 4. Equalized image

III. VASCULAR NETWORK DETECTION

Notwithstanding the great improvement of image contrast achieved by the pre-processing stage, images still present vessels with different intensities, badly defined edges and even discontinuities due to their vanishment inside of the endometrium. These facts still prevent the usage of methods like Thresholding, Edge Detection and Vessel Tracking. The developed method relies on the fact that vessels have lower intensities than the image background. Therefore an image derived by a local minimum detection operation is first obtained. Next this image is simplified through thinning. The skeleton is further decomposed in to several segments which are classified. Segments classified as vessels are finally used to reconstruct the network. All these operations are described in the following sections.

A. Local Minimum Detection

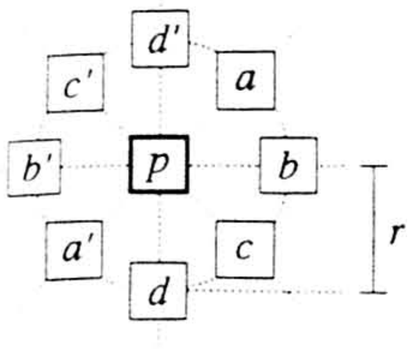
For this study, one pixel is considered a local minimum along a certain direction if its intensity is lower than the intensities of both pixels situated r pixels away from it. The smallest of the two intensity differences is designated directional contrast.



$$c_{aa'} = \min(a-p, a'-p) \qquad (3)$$

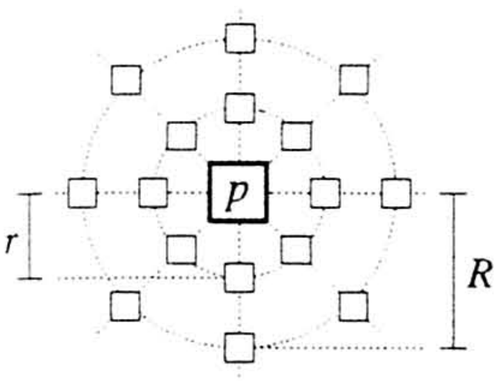
One pixel is a local minimum if it is so along at least one of four directions: horizontal, vertical and the two diagonals. The contrast will be the higher of the

directional contrasts and gives a kind of local minimum measure.



$$c_r = \max(c_{aa'}, c_{bb'}, c_{cc'}, c_{dd'}) \tag{4}$$

The final result is influenced by the choice of the r distance, radius of a ring of pixels surrounding the pixel under analysis. Small radius do not allow the detection of thick vessels because the whole ring stays inside the vessel. On the other hand large values of r can lead to a loss of some small calibre vessels. Therefore a mixed process is used with two different radius: r and R. The final contrast is the greatest of the contrasts achieved in each ring.



$$c = \max(c_r, c_R) \tag{5}$$

To avoid getting a contrast image excessively noisy the equalized image should be previously filtered through a low-pass filter. The contrast image is then thresholded at one contrast level considered adequate to get the desired local minimum image. In the developed system this level can be selected by the user due to great variability of contrast presented by the images. In Fig. 5 the local minimum image is shown. The values used for r and R were respectively 3 and 6.

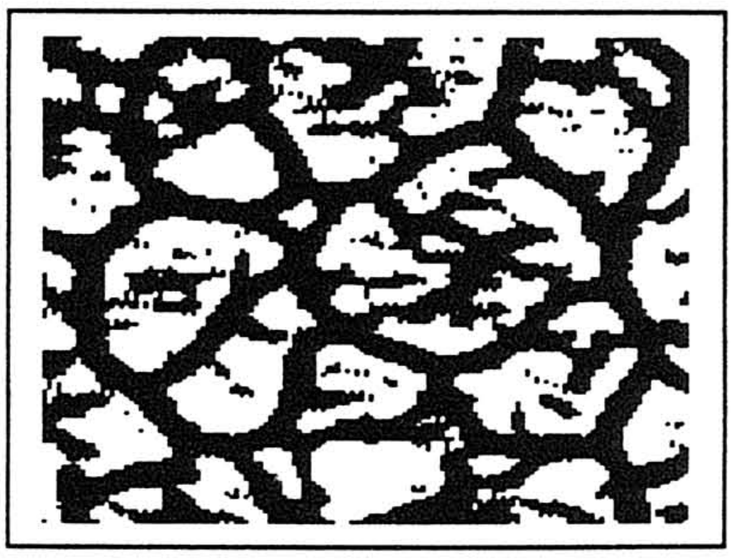


Fig. 5. Local minimum

B. Thinning

In order to obtain a simple image without changing its connectivity and topology a thinning operation is executed.

In the method we used, known as NSEW, the transition pixels from a certain direction are removed if this action does not violate the connectivity conditions of the image. To get a skeleton as much centralized as possible this removal procedure must be realised alternately from four directions: North, South, East and West.

Assuming that background pixels have value 0 and the others 1 and that n, ne, e, se, s, sw, w, and nw represent the neighbour pixels of pixel p under analysis, it is possible to establish the following definitions:

p is a transition pixel from direction x iff:

$$\begin{aligned} x &= 0 \\ p &= 1 \end{aligned} \tag{6}$$

removal of pixel p does not violate connectivity conditions iff:

$$\begin{aligned} &n + ne + e + se + s + sw + w + nw > 1 \\ &\bar{n}.ne.\bar{e} = 0 \\ &\bar{e}.se.\bar{s} = 0 \\ &\bar{s}.sw.\bar{w} = 0 \\ &\bar{w}.nw.\bar{n} = 0 \\ &\bar{n}.e.\bar{s}.w = 0 \\ &n.\bar{e}.s.\bar{w} = 0 \end{aligned} \tag{7}$$

Fig. 6 shows the skeleton of the previously determined local minimum image.

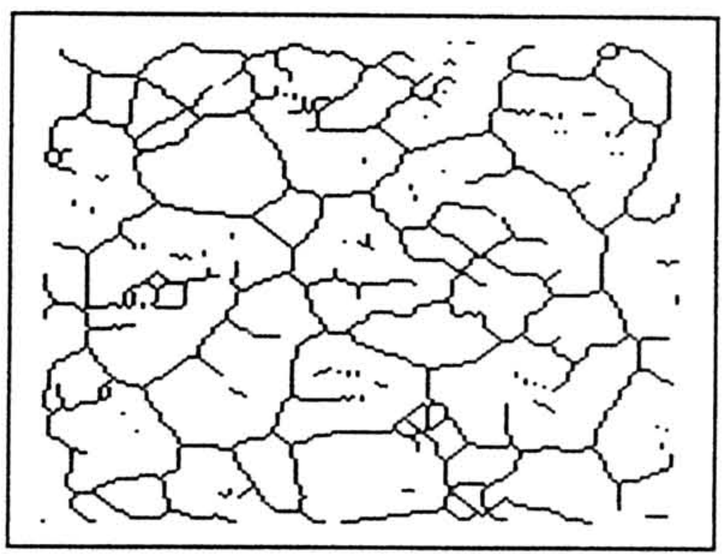
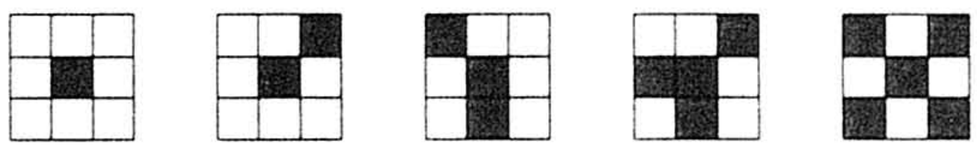


Fig. 6. Skeleton

C. Segment Breaking

Skeleton pixels have different characteristics depending on the number of neighbour pixels that are set. A pixel is isolated, terminal, passing, joining or crossing according to that number being equal to 0, 1, 2, 3 and 4. The thinning operation assures that values over 4 do not exist.



isolated pixel | *terminal pixel* | *passing pixel* | *joining pixel* | *crossing pixel*

A simple and rapid process that allow the desired decomposition of the skeleton into its several segments is the removal of all joining and crossing pixels.

D. Classification

The aim of classification stage is to choose which segments are most appropriate to constitute vessels

according to their characteristics. These characteristics are:

- average intensity
- average contrast

They can be determined by averaging the correspondent values of intensity and contrast in each pixel that belong to the segment. These values can be found respectively on the equalized and contrast images.

The classifier used is very simple since the only condition to be fulfilled in order to a certain segment being classified as a vessel is the following one:

$$(Contrast > C) \vee (Intensity < I) \qquad (8)$$

The value used for parameter I was 60. Parameter C can be selected by the user due to the already mentioned great variability of contrast presented by the images. As can be seen in Fig. 7 only some segments were classified as vessels.

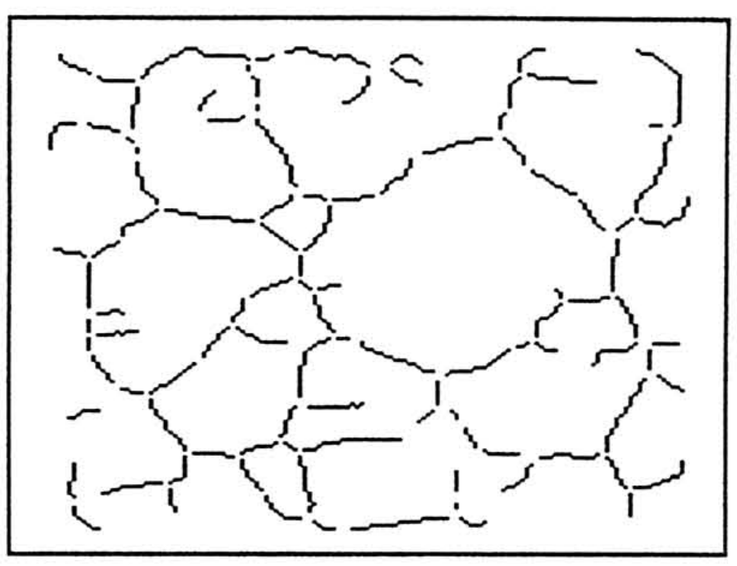


Fig. 7. Segments classified as vessels

E. Vascular Network Reconstruction

Once the vessels located it is necessary to reconstruct the vascular network. To do so, all the pixels in those segments classified as vessels are considered to be center of a circle with a diameter equal to the vessel calibre in that location. The successive placement of circles will originate the desired network. Edge smoothing is performed by network dilatation followed by erosion.

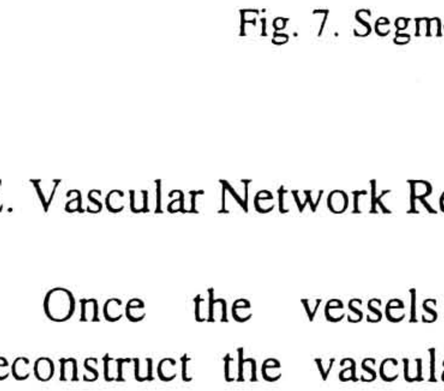


Fig. 8. Reconstruction method through placement of circles

Calibre determination is done by an analysis of the intensity profile along the direction of higher contrast. Since calibre is the distance between the vessel edges it is necessary to locate them. Among several criterions that could be used to perform this location we selected one that establishes that an edge is situated at the crossing point of the profile with intensity m, where:

$$m = \frac{\max_{int} + \min_{int}}{2} \qquad (9)$$

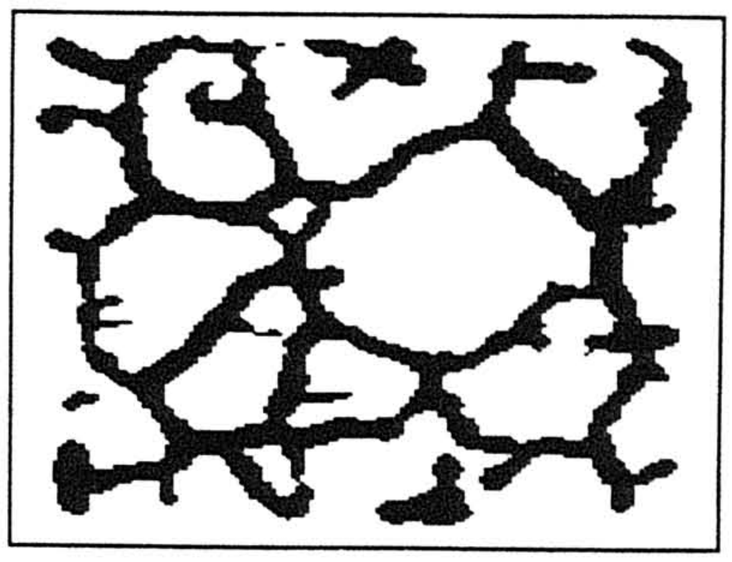


Fig. 9. Final image of the uterine vascular network

IV. FEATURES EXTRACTION

Once accomplished the detection of the uterine vascular network it is necessary to extract several features to characterise it from a clinical point of view. The chosen features are as follows:

F1 vascular area
F2 vascular perimeter
F3 number of segments
F4 number of nodes
F5 minimum calibre
F6 average calibre
F7 maximum calibre
F8 minimum length
F9 average length
F10 maximum length
F11 area homogeneity
F12 perimeter homogeneity
F13 calibre homogeneity
F14 nodes homogeneity

Note that the first four features are given in a percentage form due the inconstancy of ROI size. Perimeter is established by filtering the final image through an edge detector (Laplacian).

One node corresponds to a place of convergence of three or more segments. This place is identified by the presence inside a certain neighbourhood of three or more terminal pixels.

Finally, homogeneities attempt to represent the uniformity in the distribution of the network along the region of interest.

V. SYSTEM

The hardware is composed by a personal computer with an image acquisition board. The input of this board is connected to a video cassette recorder that reproduces the images previously recorded during standard diagnostic procedures. The output is connected to an RGB display for visualisation of those images as well as images computed during the processing stages.

The software package SPIMUT developed in C language executes the processing algorithms described in this paper. This software also deals with acquisition, loading and recording of images as well as storage on disk of the computed features for subsequent analysis.



Fig. 10. System

VI. RESULTS AND CONCLUSIONS

The system is presently being validated at the Gynaecology Service of S. João Hospital, Porto. It is already possible to confirm the good results obtained even when the images are adverse as illustrated in the following two other cases.

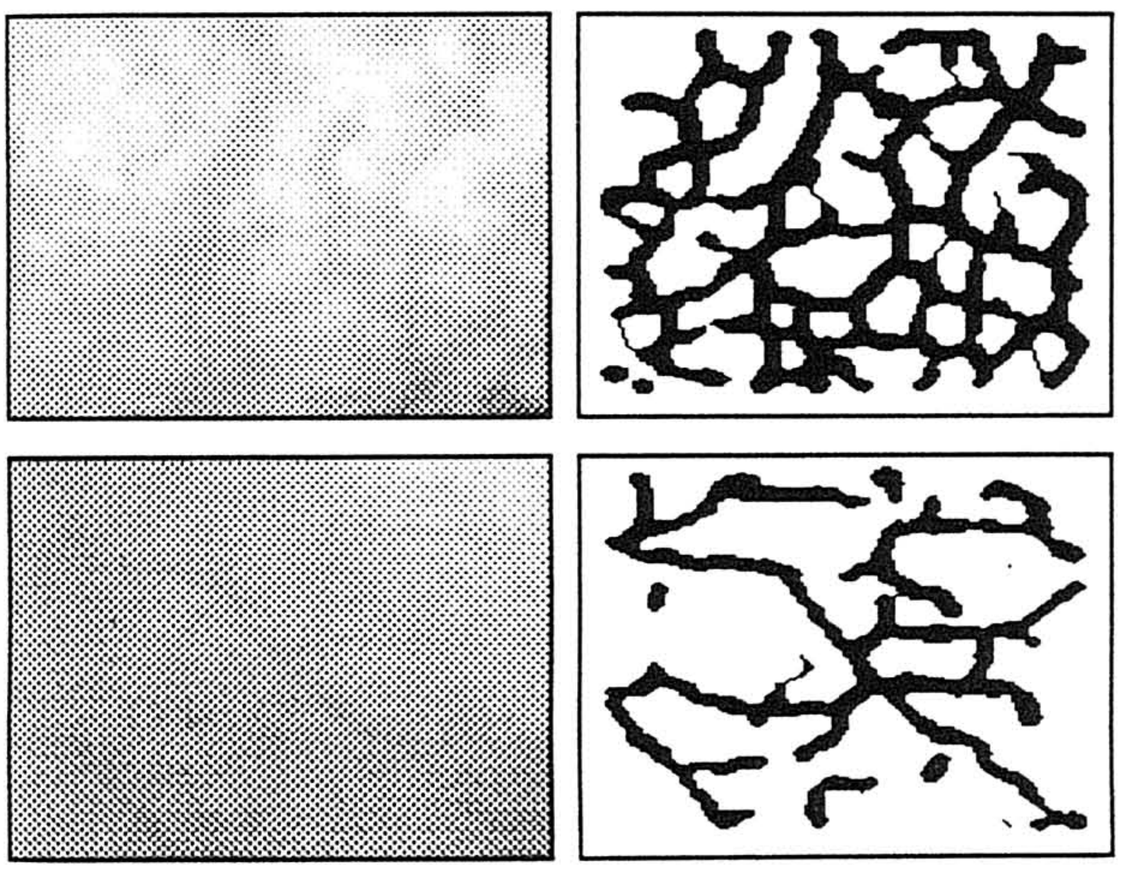


Fig. 11. Some other cases

The main characteristics of the developed algorithm in opposition with other vascular network detection methods are:

- *processing speed* - in a 486DX computer at 33 MHz a 256 x 256 pixels ROI is entirely processed and the features extracted in about 15 seconds.
- *local and global approach* - operations like minimum detection are local (at pixel's level); by the contrary the classification has a global scope (at segment's level).
- *image quality demand* - this algorithm is not very exigent in terms of vessel edges definition as well as continuity of those same vessels.
- *flexibility* - the adjustment of some parameters (r, R, I, C) allows the adaptation of the algorithm to other type of images.

In conclusion, it is our expectation that this system will contribute to a better knowledge of the uterus functioning through a more reliable, precise and objective analysis of uterine hysteroscopic images. New insights on diagnostic, namely in the field of infertility, are expected to be feasible.

VII. REFERENCES

[1] Gray Level Thresholding in Badly Illuminated Images, J. R. Parker, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13, No. 8, August 1991, pp 813-819.

[2] Detection of Blood Vessels in Retinal Images Using Two-Dimensional Matched Filters, S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, M. Goldbaum, IEEE Transactions on Medical Imaging, Vol. 8, No 3, September 1989, pp 263-269.

[3] An Artificial Vision System for X-Ray Images of Human Coronary Trees, G. Coppini, M. Demi, G. Valli, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, No. 2, February 1993, pp 156-162.

[4] Automated Identification of Vessels Contours in Coronary Arteriograms by an Adaptative Tracking Algorithm, Ying Sun, IEEE Transactions on Medical Imaging, Vol. 8, No 1, March 1989, pp 78-88.

[5] Automated Tracking of the Vascular Tree in DSA Images Using a Double-Square-Box Region-of-Search Algorithm, K. R. Hoffmann, Kunio Doi, Heang-Ping Chan, L. Fencil, H. Fujita, A. Muraki, SPIE Vol. 626, Medicine XIV/PACS IV, 1986, pp 326-333.

[6] Baseline Fluctuations Removal from Impedance Cardiographic Signals, H. M. Mendonça, J. Jossinet, J. P. Marques de Sá, Innovation et Technologie en Biologie et Medicine, Vol. 13, No. 2, 1992, pp 183-187.


Disquete de demonstração
do programa
Spinut
ATENÇÃO:
ca de aquisição
liar.

FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO
BIBLIOTECA
0000002221