

AnisotropicDiffusion (1 call, 0.422 sec)

Generated 09-May-2007 23:20:39 using real time.

M-function in file [H:\edgy\Implementering\profilekoder\artikel1\AnisotropicDiffusion.m](#)
[[Copy to new window for comparing multiple runs](#)]

Refresh

- ☐ Show parent functions
- ☒ Show busy lines
- ☒ Show child functions
- ☐ Show M-Lint results
- ☒ Show file coverage
- ☒ Show function listing

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
100	[a, b, c, d] = fastD(J, alpha,...	2	0.156 s	37.0%	<div></div>
96	J = imfilter(imfilter(nablaI,K...	2	0.155 s	36.7%	<div></div>
104	I = I + divergenceC(a.*I_x + b...	2	0.032 s	7.6%	<div></div>
86	I_y=imfilter(I,derivKernel', '...	2	0.031 s	7.3%	<div></div>
87	I_x=imfilter(I,derivKernel, 'c...	2	0.016 s	3.8%	<div></div>
Other lines & overhead			0.032 s	7.6%	<div></div>
Totals			0.422 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
imfilter	M-function	8	0.202 s	47.9%	<div></div>
fastD	MEX-function	2	0.140 s	33.2%	<div></div>
divergenceC	MEX-function	2	0.032 s	7.6%	<div></div>
IsScalar	M-function	8	0 s	0%	
IsPositiveInteger	M-function	1	0 s	0%	
createDerivKernel	M-function	1	0 s	0%	

createGaussKernel	M-function	1	0 s	0%	
Self time (built-ins, overhead, etc.)			0.048 s	11.4%	■
Totals			0.422 s	100%	

Coverage results

[[Show coverage for parent directory](#)]

Total lines in file	138
Non-code lines (comments, blank lines)	62
Code lines (lines that can run)	76
Code lines that did run	32
Code lines that did not run	44
Coverage (did run/can run)	42.11 %

Function listing

Color highlight code according to

time

```
time  calls  line
      1 function [I, timeSpent] = AnisotropicDiffusion( inIma
      2         sigma2, delta2, s2, alpha, beta, tau, numIter, us
      3 %AnisotropicDiffusion Implements Nonlinear Anisotropic
      4 % [I, timeSpent] = AnisotropicDiffusion( inImage, s
      5 %   sigma2, delta2, s2, alpha, beta, tau, useAOS, num
      6 %
      7 % Returns:
      8 % I          Processed Image after selected iterat
      9 % timeSpent  Time spent processing exclusive creat
     10 %
     11 % Parameters:
     12 % inImage    Image to process. Must be gray scale
     13 % sigma1     Sigma for derivating kernel
     14 % delta1     Cut of tail of derivating kernel at t
     15 % sigma2     Sigma for low pass kernel
     16 % delta2     Cut of tail of low pass kernel at thi
     17 % s2        Stop value for diffusion in gradient
     18 % alpha     Isotropic diffusion number
     19 % beta      Maximum amount of diffusion. Small be
     20 % tau       Time step
     21 % numIter   Maximum number
     22 % useAOS    Choose to use an Additive Operatorsspl
     23 %           of a divergence operator
     24 % graphs    1 - Show graphs, 0 - Show no graphs
     25 %
     26 % Requirements IMFILTER and IMSHOW in Image Process
     27 % See also IMFILTER, IMSHOW.
     28 %
     29 % Licensed under BSD as a part of EDGY project sour
     30 % see readme.txt file. For more information see pro
     31 %
```

```

32 % Revision: 1.0
33 % Date: 2007/05/09
34 % Author: Henrik Wolkesson
35
1 36 if ~IsScalar(sigma1) || sigma1 < 0 || ...
37     ~IsScalar(delta1) || delta1 < 0 || ...
38     ~IsScalar(sigma2) || sigma2 < 0 || ...
39     ~IsScalar(delta2) || delta2 < 0
40     error('Sigma1, Sigma2, Delta1 and Delta2 must be
41 end
1 42 if ~IsScalar(s2) || s2 < 0 || ~IsScalar(alpha) || alp
43     ~IsScalar(beta) || beta < 0 || ~IsScalar(tau)
44     error('s2, alpha, beta and tau must be a positive
45 end
1 46 if ~IsPositiveInteger(numIter)
47     error('numIter must be positive integer');
48 end
1 49 if useAOS < 0 || useAOS > 1 || round(useAOS) ~= useAC
50     error('useAOS must be either 0 or 1');
51 end
1 52 if graphs < 0 || graphs > 1 || round(graphs) ~= graph
53     error('Graphs must be either 0 or 1');
54 end
55
56 % Section 1: Initize
1 57 inImage = double(inImage);
1 58 I=inImage;
< 0.01 1 59 N = size(I,1);
< 0.01 1 60 M = size(I,2);
0.02 1 61 nablaI=zeros(N,M,2);
1 62 if (useAOS == 1)
63     unitMatrixM = sparse(eye(M));
64     unitMatrixN = sparse(eye(N));
65 end
66
67 % Section 2: Create kernels
1 68 derivKernel = createDerivKernel(sigma1, delta1);
1 69 Kx = createGaussKernel(sigma2,delta2);
70
1 71 if (graphs == 1)
72     figure('Name', 'Kernels');
73     subplot(1,2,1);plot(Kx);
74     title(['Gauss Kernel (Size: ' num2str(size(Kx,2))
75     subplot(1,2,2);plot(derivKernel);
76     title(['Derivating Kernel (Size: ' num2str(size(d
77 end;
78
1 79 if (size(Kx,2)>size(I,2))
80     warning('Derivating kernel is bigger than image')
81 end
82
< 0.01 1 83 tic;
1 84 for k = 1:numIter
85     % Section 3: Calculate derivatives
0.03 2 86 I_y=imfilter(I,derivKernel', 'conv', 'replicate')
0.02 2 87 I_x=imfilter(I,derivKernel, 'conv', 'replicate');
88

```

```

89 % Section 4: Create structure matrix
0.02 2 90 nablaI(:,:,1)=I_x.*I_x;
2 91 nablaI(:,:,2)=I_x.*I_y;
2 92 nablaI(:,:,3)=I_y.*I_y;
93
94 % Section 5: LP filter structure matrix
0.16 2 95 Ky = Kx';
2 96 J = imfilter(imfilter(nablaI,Kx, 'conv', 'replicat
97 Ky, 'conv', 'replicate'));
98
99 % Section 6-8: Eigenanalays, modulation and const
0.16 2 100 [a, b, c, d] = fastI(J, alpha, beta, s2, inImage,I
101
102 % Section 9: Solve diffusion ekvation
0.03 2 103 if (useAOS == 0)
2 104 I = I + divergence(a.*I_x + b.*I_y, c.*I_x +
105 else
106 InextN = solveAOSN(I, N, M, a, b, c, d, tau,
107 InextM = solveAOSM(I, N, M, a, b, c, d, tau,
108 I = InextM + InextN;
109 end
110
111 % Optional plotting
2 112 if (graphs == 1)
113 figure;
114 subplot(3,1,1);imshow(I,[]);title(['Iteration
115 title(['I of iterataion ' num2str(k)]);
116 subplot(3,1,2);imshow(I_x, []);
117 title(['Derivate dI/dx of iterataion ' num2st
118 subplot(3,1,3);imshow(I_y, []);
119 title(['Derivate dI/dy of iterataion ' num2st
120 figure('Name', ['LP-filtered Tensor J of itera
121 subplot(3,1,1);imshow(J(:,:,1),[]);
122 title('dI/dx*dI/dx');
123 subplot(3,1,2);imshow(J(:,:,2),[]);
124 title('dI/dx*dI/dy');
125 subplot(3,1,3);imshow(J(:,:,3),[]);
126 title('dI/dy*dI/dy');
127 end
2 128 end
129
130 % Optional Plotting
1 131 if (graphs == 1)
132 figure;plot(inImage(floor(N/2),:),'r');hold on; p
133 legend('Original with Speckle','Processed');
134 end
135
136 % Section 10: Return time and processed image
< 0.01 1 137 timeSpent = toc;
1 138 I = uint8(I);

```