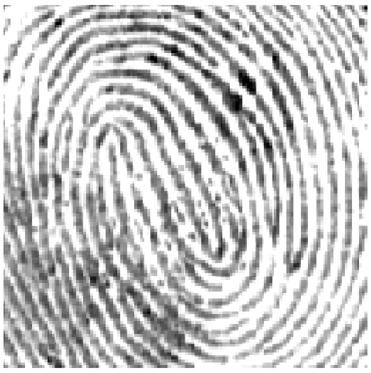
Differential Equations in Image Processing and Computer Vision 2008
Example Solutions for Programming Assignments 3 (P3)

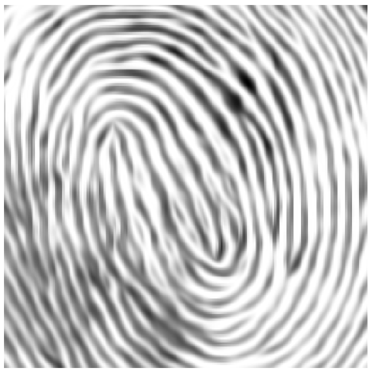
Problem 1

- The complete file `diff_tensor.c` with supplemented code reads

```
/* ------------------------------------------------------------ */
void diff_tensor

     (float    C,         /* contrast parameter */
      float    alpha,     /* linear diffusion fraction */
      long     nx,        /* image dimension in x direction */
      long     ny,        /* image dimension in y direction */
      float    **dxx,     /* in: ST element / out DT element */
      float    **dxy,     /* in: ST element / out DT element */
      float    **dyy)     /* in: ST element / out DT element */

/* Calculates the diffusion tensor of CED using the structure tensor. */

{
long    i, j;           /* loop variables */
float   beta;           /* timesaver */
float   c, s;           /* specify first eigenvector */
float   mu1, mu2;       /* eigenvalues of structure tensor */
float   lam1, lam2;     /* eigenvalues of diffusion tensor */

beta = 1.0 - alpha;

for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     {
     /* principal axis transformation */
     PA_trans (dxx[i][j], dxy[i][j], dyy[i][j], &c, &s, &mu1, &mu2);

     /* calculate eigenvalues */
     lam1 = alpha;
     if (mu1 == mu2)
        lam2 = alpha;
     else
        lam2 = alpha + beta * exp (- C / ((mu1 - mu2) * (mu1 - mu2)));

     /* principal axis backtransformation */
     PA_backtrans (c, s, lam1, lam2, &dxx[i][j], &dxy[i][j], &dyy[i][j]);
     }
return;
}  /* diff_tensor */
/* ------------------------------------------------------------ */
```

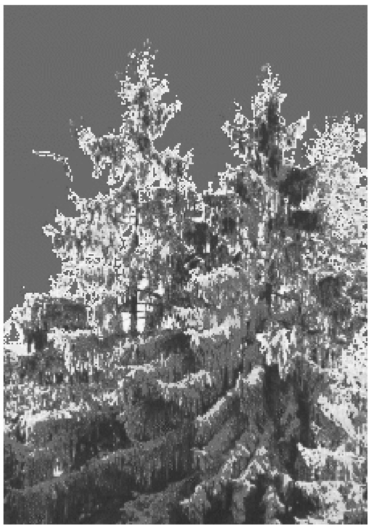
- If we use 40 iterations with the parameters $C = 1$, $\sigma = 0.5$, $\rho = 4$, $\alpha = 0.001$, $\tau = 0.2$, we obtain the following result

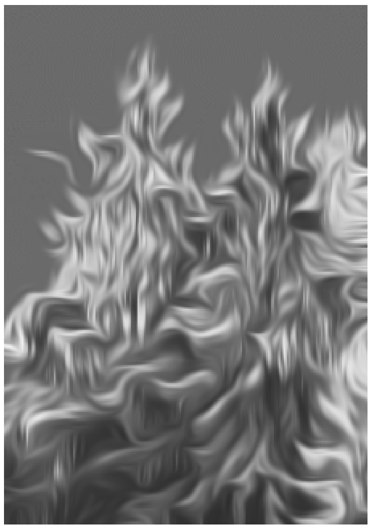
  
  original

  
  CED

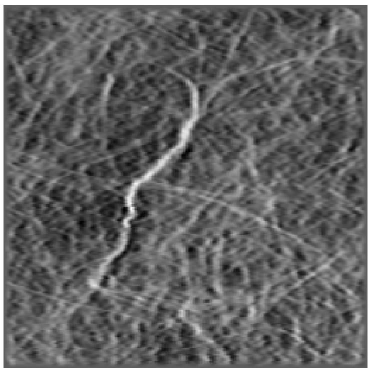
  During the iterations one can clearly see that the maximum-minimum-principle is violated. The minimum becomes negative and the maximum clearly exceeds the maximum of the original image given by 255.

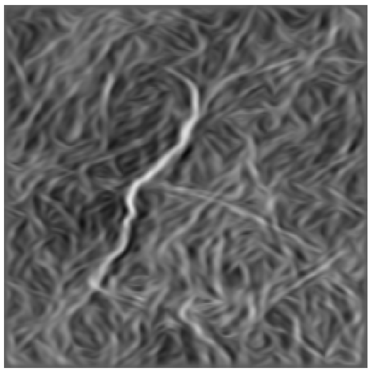
- In order to create some nice christmas postcards we apply the same parameters as for the fingerprint. The obtained result is given by

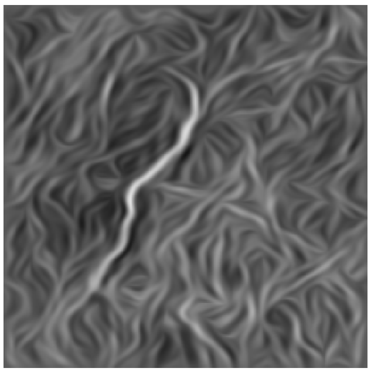
  
  original

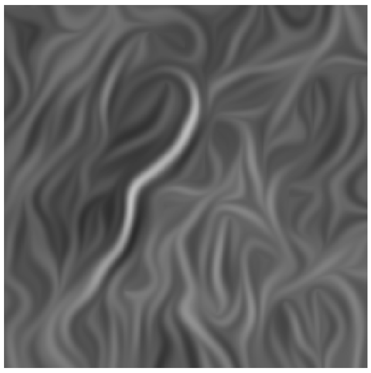
  
  CED

- One can also use the CED to visualise the stripes from the image `fabric.pgm` on different scales. To this end, one has to use different diffusion times, i.e. different numbers of iterations

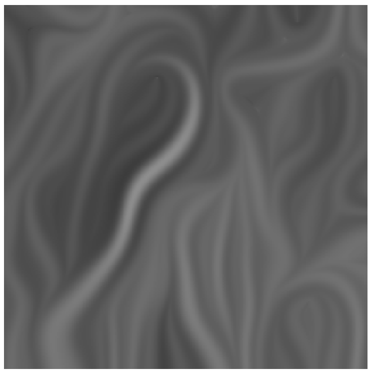

original


CED $t = 2$


CED $t = 8$


CED $t = 80$

CED $t = 200$


CED $t = 800$

Problem 2

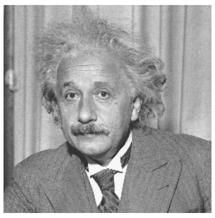
- The complete routine `isonodiff` with supplemented code reads

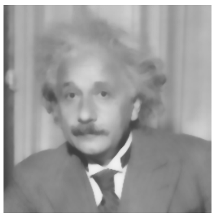
```
/* ------------------------------------------------------ */
void isonondiff

     (float    ht,          /* time step size, 0 < ht <= 0.25 */
      long     nx,          /* image dimension in x direction */
      long     ny,          /* image dimension in y direction */
      float    hx,          /* pixel size in x direction */
      float    hy,          /* pixel size in y direction */
      float    alpha,       /* weight of the similarity term */
      float    lambda,      /* contrast parameter */
      float    **f,         /* initial image, unchanged */
      float    **u)         /* input: original image;  */
                            /* output: smoothed */


/*
 Nonlinear diffusion / regularisation with Charbonnier
 diffusivity. Modified explicit scheme.
*/

{
long    i, j;             /* loop variables */
float   rxx, ryy;         /* time savers */
float   **v;              /* work copy of u */
float   **dc;             /* diffusion coefficient */
float   dv_dx, dv_dy;     /* derivatives */
float   two_hx, two_hy;   /* time savers */
float   grad_sqr;         /* |grad(u)|^2 */
float   help1, help2;     /* time saver */


/* ---- allocate storage ---- */
alloc_matrix (&v,  nx+2, ny+2);
alloc_matrix (&dc, nx+2, ny+2);


/* ---- copy u into f ---- */
for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     v[i][j] = u[i][j];


/* ---- calculate diffusivity ---- */
two_hx = 2.0 * hx;
two_hy = 2.0 * hy;
help1  = 1.0 / (lambda * lambda);

/* dummy boundaries */
dummies (v, nx, ny);

for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     {
     /* calculate grad_sqr */
     dv_dx = (v[i+1][j] - v[i-1][j]) / two_hx;
     dv_dy = (v[i][j+1] - v[i][j-1]) / two_hy;
     grad_sqr = dv_dx * dv_dx + dv_dy * dv_dy;

     /* calculate diffusivity dc */
     dc[i][j] = 1.0 / sqrt(1.0 + help1 * grad_sqr);
     }


/* ---- calculate explicit nonlinear diffusion of u ---- */
/* dummy boundaries */
dummies (dc, nx, ny);

/* diffuse */
rxx = ht / (2.0 * hx * hx);
ryy = ht / (2.0 * hy * hy);
for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     u[i][j] = v[i][j]
        + rxx * ( (dc[i+1][j] + dc[i][j]) * (v[i+1][j] - v[i][j])
                + (dc[i-1][j] + dc[i][j]) * (v[i-1][j] - v[i][j]) )
        + ryy * ( (dc[i][j+1] + dc[i][j]) * (v[i][j+1] - v[i][j])
                + (dc[i][j-1] + dc[i][j]) * (v[i][j-1] - v[i][j]) );


/* ---- blend with the original image ---- */
for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
        u[i][j] = (alpha*u[i][j] + ht*f[i][j])/(alpha+ht);


/* ---- disallocate storage ---- */
disalloc_matrix (v,  nx+2, ny+2);
disalloc_matrix (dc, nx+2, ny+2);

return;

} /* isonondiff */
/* ---------------------------------------------------------- */
```

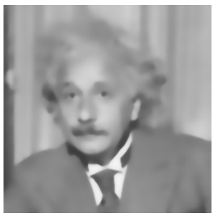
- In order to compare the diffusion-reaction with the pure diffusion method we have to recall three things:
  - The pure diffusion method can be obtained from the diffusion-reaction method for $\alpha \to \infty$, since for a very large value of α the diffusion term dominates the reaction term.
  - The diffusion time t and the time step size τ have a totally different meaning in both approaches: For the diffusion method the total diffusion time is given by $t = n_{\text{iter}} \cdot \tau$. Here, the diffusion time determines the impact of the filter. In contrast, in the case of the diffusion-reaction method, the time t and the corresponding time step size τ are only *pure numerical* parameters: Here, we are interested in the *steady state*, i.e. in the state for $t \to \infty$. Therefore, we have to a large number of iterations in the case of the diffusion-reaction method. In general, one stops the iteration scheme when the change from one iteration to the next is rather small (e.g. one can use a threshold for the variance as stopping criterion).
  - In the case of the diffusion reaction method the impact of the filter is steered by the value of the parameter α. Its role is comparable to that of the diffusion time in the case of the diffusion filter.

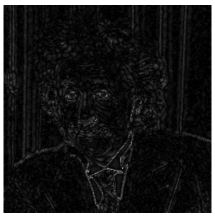
In our first experiment we compare the diffusion-reaction and the pure diffusion method (e.g. $\alpha = 100000$) for the `einstein.pgm` image with parameter $\lambda = 4$. For the diffusion-reaction method we chose $\underline{\alpha = 10}$ and approximated the steady state with 500 iterations of step size $\tau = 0.2$. For the pure diffusion method this corresponds to a diffusion time of $\underline{t = 10}$ which can for instance be achieved by performing 50 iterations with $\tau = 0.2$.

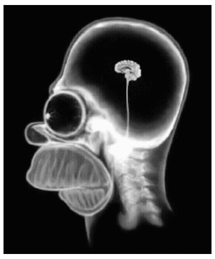
In our second experiment we compare the diffusion-reaction and the pure diffusion method ($\alpha = 100000$) for the `sbrain.pgm` image with parameter $\lambda = 5$. For the diffusion-reaction method we chose $\underline{\alpha = 20}$ and approximated the steady state with 600 iterations of step size $\tau = 0.2$. For the pure diffusion method this corresponds to a diffusion time of $\underline{t = 20}$ which can for instance be achieved by performing 100 iterations with $\tau = 0.2$.

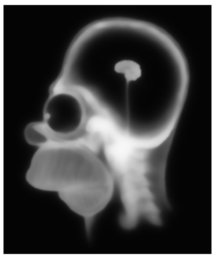

original

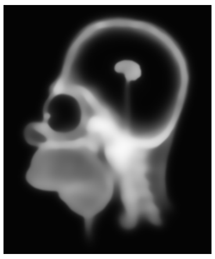

diffusion-reaction

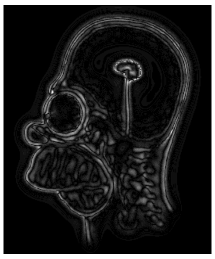

diffusion


difference


original


diffusion-reaction


diffusion


difference

In both cases one can observe differences between the two methods. This, however, is not surprising, since both methods are closely related but nevertheless *different* techniques for the denoising of images.