Differential Equations in Image Processing and Computer Vision 2008
Example Solutions for Programming Assignments 5 (P5)

Problem 1

- In the file `optic.c` the routine `flow` had to be supplemented with code for the modified explicit scheme. The complete routine `flow` reads

```
/* ------------------------------------------------------ */
void flow

     (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    **fx,        /* x derivative of image */
      float    **fy,        /* y derivative of image */
      float    **ft,        /* theta derivative of image */
      float    alpha,       /* smoothness weight */
      float    lambda,      /* contrast parameter */
      float    **u,         /* x component of optic flow */
      float    **v)         /* v component of optic flow */


/*
 Optic flow iteration by regarding it as a coupled system of
 two nonlinear diffusion-reaction equations.
 Isotropic nonlinear diffusion with explicit discretisation.
*/

{
long    i, j;             /* loop variables */
float   rx, ry;           /* mesh ratios, time savers */
float   **u1, **v1;       /* intermediate results */
float   **dc;             /* diffusion coefficient */
float   **rhs;            /* right hand side of linear system */
float   **diag1;          /* summand of diagonal part of system matrix */
float   ux, uy, vx, vy;   /* derivatives */
float   help, rxx, ryy;   /* time savers */
float   two_hx, two_hy;   /* time savers */
float   grad_sqr;         /* |grad(v)|^2 */
float   lambda_sqr;       /* lambda^2 */
float   n,s,w,e,c;

/* ---- allocate storage ---- */

alloc_matrix (&u1, nx+2, ny+2);
alloc_matrix (&v1, nx+2, ny+2);
alloc_matrix (&dc, nx+2, ny+2);
alloc_matrix (&rhs, nx+2, ny+2);
alloc_matrix (&diag1, nx+2, ny+2);


/* ---- copy u, v into u1, v1 ---- */

for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     {
     u1[i][j] = u[i][j];
```

```
     v1[i][j] = v[i][j];
     }


two_hx = 2.0 * hx;
two_hy = 2.0 * hy;
lambda_sqr = lambda * lambda;
dummies (u1, nx, ny);
dummies (v1, nx, ny);

for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     {
     /* calculate grad(f) */
     ux = (u1[i+1][j] - u1[i-1][j]) / two_hx;
     uy = (u1[i][j+1] - u1[i][j-1]) / two_hy;
     vx = (v1[i+1][j] - v1[i-1][j]) / two_hx;
     vy = (v1[i][j+1] - v1[i][j-1]) / two_hy;
     grad_sqr = ux * ux + uy * uy + vx * vx + vy * vy;
     dc[i][j] = 1.0 / sqrt (1.0 + grad_sqr / lambda_sqr);
     }

dummies (dc, nx, ny);
/* ---- calculate diffusion of the first equation ---- */

help = ht / alpha;
rxx  = ht / (2.0 * hx * hx);
ryy  = ht / (2.0 * hy * hy);

/* dummy boundaries */
dummies (u1, nx, ny);

for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
 {
   /* SUPPLEMENT CODE */

   /* weights of neighbour pixels */
   w=(dc[i-1][j  ]+dc[i][j])*rxx;
   n=(dc[i  ][j-1]+dc[i][j])*ryy;
   s=(dc[i  ][j+1]+dc[i][j])*ryy;
   e=(dc[i+1][j  ]+dc[i][j])*rxx;

   /* weight of centre pixel*/
   c=-(w+n+s+e);

   /* compute modified explicit step */
   u[i][j] = (u1[i][j] + w * u1[i-1][j  ]
              + n * u1[i  ][j-1]
              + c * u1[i  ][j  ]
              + s * u1[i  ][j+1]
              + e * u1[i+1][j  ]
                    - help*fx[i][j]*(fy[i][j]*v1[i][j]+ft[i][j]))
     /(1.0+help*fx[i][j]*fx[i][j]);
 }

/* ---- calculate diffusion of the second equation ---- */

/* dummy boundaries */
dummies (v1, nx, ny);

for (i=1; i<=nx; i++)
```

```
for (j=1; j<=ny; j++)
{
  /* SUPPLEMENT CODE */

  /* weights of neighbour pixels */
  w=(dc[i-1][j  ]+dc[i][j])*rxx;
  n=(dc[i  ][j-1]+dc[i][j])*ryy;
  s=(dc[i  ][j+1]+dc[i][j])*ryy;
  e=(dc[i+1][j  ]+dc[i][j])*rxx;

  /* weight of central pixel */
  c=-(w+n+s+e);

  /* compute modified explicit step */
  v[i][j] = (v1[i][j] + w * v1[i-1][j  ]
              + n * v1[i  ][j-1]
              + c * v1[i  ][j  ]
              + s * v1[i  ][j+1]
              + e * v1[i+1][j  ]
                      - help*fy[i][j]*(fx[i][j]*u1[i][j]+ft[i][j]))
    /(1.0+help*fy[i][j]*fy[i][j]);
}


/* ---- disallocate storage ---- */

disalloc_matrix (u1, nx+2, ny+2);
disalloc_matrix (v1, nx+2, ny+2);
disalloc_matrix (dc, nx+2, ny+2);
disalloc_matrix (rhs, nx+2, ny+2);
disalloc_matrix (diag1, nx+2, ny+2);
return;

} /* flow */

/* ------------------------------------------------------------ */
```

- Moreover, the meaning of the parameters α and σ as well as the number of iterations had to be investigated. To this end, experiments with the image pair `pig1.pgm` and `pig2.pgm` have to be performed.

`pig1.pgm`

`pig2.pgm`

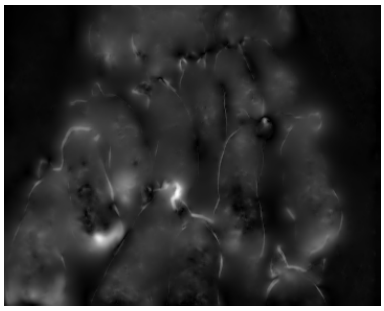
In all experiments we chose $\tau = 0.2$. Let us start with the evaluation of the results for different values of the **smoothness weight** α. As one can

see the flow field becomes smoother if this parameter is increased. If α is chosen sufficiently large the flow can be estimated for the complete pigs and not just for their hull (where the image gradient are large).

The second parameter, the parameter λ is the so-called **contrast parameter**. While a small value allows the preservation of small details (edges with a small magnitude are preserved), a very large value corresponds basically to homogeneous diffusion (even very large edges are not preserved any longer). This can clearly been seen from the corresponding flow fields.

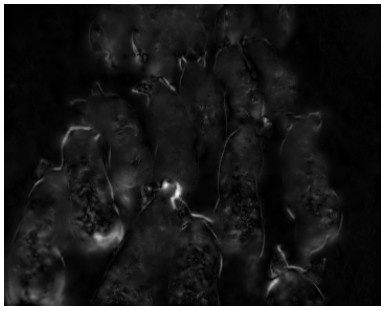
The third (and the only numerical parameter) are the **number of iteration steps**. For a very large number of iterations the results achieve the smoothness specified by the value for α (we are interested in the steady state for $t \to \infty$). For a small number of iterations, this degree of smoothness is not reached yet and thus the obtained flow field look similar to the results for a much smaller value of α.

$\alpha = \mathbf{1}$, $\lambda = 5$, $n = 1000$

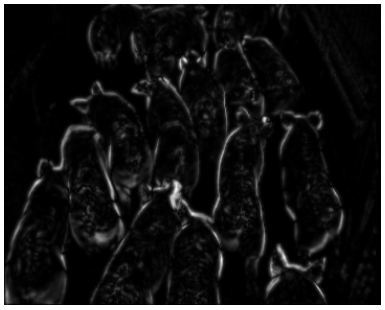


$\alpha = \mathbf{1000}$, $\lambda = 5$, $n = 1000$

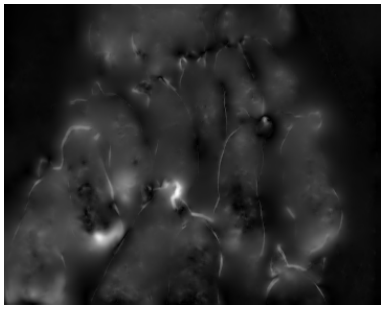
$\alpha = 100$, $\lambda = \mathbf{0.1}$, $n = 1000$



$\alpha = 100$, $\lambda = \mathbf{10000}$, $n = 1000$



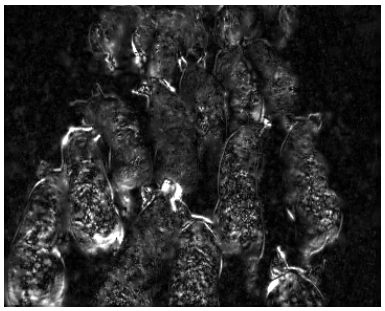
$\alpha = 1000$, $\lambda = 5$, $\mathbf{n = 10}$



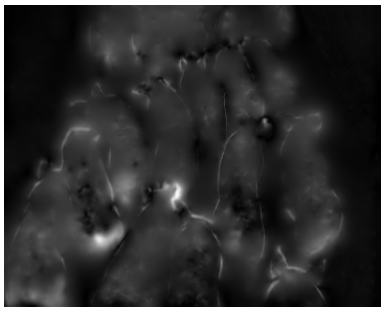
$\alpha = 1000$, $\lambda = 5$, $\mathbf{n = 10000}$

- In the second task we had to modify the program in such a way that it uses homogeneous regularisation instead of the isotropic one. This is rather simple, since for homogeneous regularisation we have $\Psi(s^2) = s^2$ and thus $\Psi'(s^2) = \frac{\mathrm{d}}{\mathrm{d}s^2} s^2 = 1$. Evidently, for the homogeneous regulariser we have to set all diffusivities to 1. While this is the correct implementation, a simple approximation can be obtained by keeping the Charbonnier function and setting the value for the contrast parameter λ very large. This is possible, since $\lim_{\lambda \to \infty} (\frac{1}{\sqrt{1+\frac{s^2}{\lambda^2}}}) = 1$.

  Two results for different values of α are shown below. One can see that the results are still rather sharp for small values of the smoothness weight. For lager values, however, the results is fairly blurry.



$\alpha = 10$, $n = 1000$



$\alpha = 1000$, $n = 1000$