Differential Equations in Image Processing and Computer Vision 2008

Example Solutions for Programming Assignments 4 (P4)

Problem 1

- In the file inpainting.c the routines `diffusion_tensor` and `inpaint` had to be supplemented with additional code. The complete routine `diff_tensor` is given by

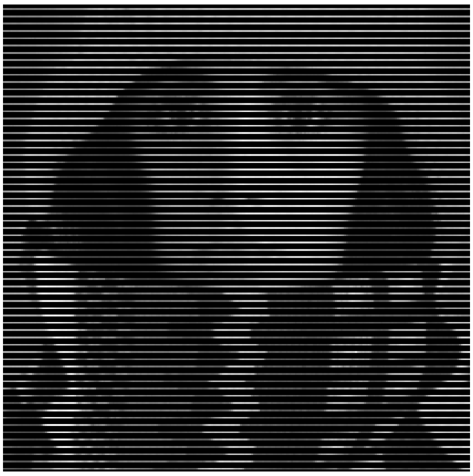
```
/* ------------------------------------------------------------ */
void diff_tensor

   (float    **v,      /* in: regularized image, unchanged */
    float    lambda,   /* contrast parameter */
    long     nx,       /* image dimension in x direction */
    long     ny,       /* image dimension in y direction */
    float    hx,       /* pixel width in x directopn */
    float    hy,       /* pixel width in y directopn */
    float    **dxx,    /* out: diffusion tensor element */
    float    **dxy,    /* out: diffusion tensor element */
    float    **dyy)    /* out: diffusion tensor element */

/*
 Calculates the diffusion tensor of the EED.
*/

{
long    i, j;              /* loop variables */
float   dv_dx, dv_dy;      /* derivatives of v */
float   two_hx, two_hy;    /* time savers */
float   c, s;              /* specify first eigenvector */
float   grad;              /* |grad(v)| */
float   mu1, mu2;          /* eigenvalues of structure tensor */
float   lam1, lam2;        /* eigenvalues of diffusion tensor */

/* time saver variables */
two_hx = 2.0 * hx;
two_hy = 2.0 * hy;
dummies (v, nx, ny);

/* for each pixel */
for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
     {
       /* calculate grad(v) */
       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  = sqrt (dv_dx * dv_dx + dv_dy * dv_dy);

       /* set up original diffusion tensor */
       dxx[i][j]=dv_dx*dv_dx;
       dxy[i][j]=dv_dx*dv_dy;
       dyy[i][j]=dv_dy*dv_dy;

       /* principal axis transformation of the diffusion tensor*/
       PA_trans(dxx[i][j], dxy[i][j], dyy[i][j],
                &c, &s, &mu1, &mu2);

       /* calculate eigenvalues lam1, lam2 of diffusion tensor */
       lam1 = 1 / sqrt(1 + grad*grad / (lambda*lambda));
       lam2 = 1;

       /* principal axis backtransformation of diffusion tensor */
       PA_backtrans(c, s, lam1, lam2,
                    &dxx[i][j], &dxy[i][j], &dyy[i][j]);
     }

 return;

}  /* diff_tensor */
/* ---------------------------------------------------------- */
```

while the complete routine `inpaint` reads

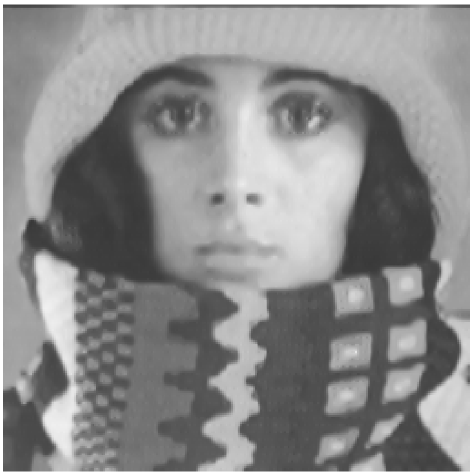
```
/* ---------------------------------------------------------- */
void inpaint

    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    lambda,      /* contrast parameter */
    float    sigma,       /* noise scale */
    float    **a,         /* inpainting mask */
    float    **u)         /* input: original image */
                          /* output: smoothed */


/*
 Impainting using an explicit evolution.
*/

{
long    i, j;                  /* loop variables */
```

```
float   **v;                   /* work copy of u */


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


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

/* ---- evolve v by edge-enhancing aniso. diffusion ---- */
eed (ht, nx, ny, hx, hy, lambda, sigma, v);

/* ---- compute inpainting ---- */
for (i=1; i<=nx; i++)
 for (j=1; j<=ny; j++)
  if (!a[i][j]) u[i][j] = v[i][j];

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

return;

} /* inpaint */
/* ------------------------------------------------------- */
```

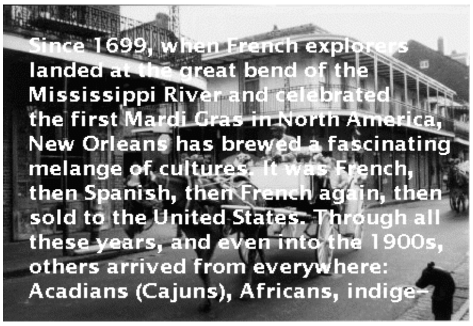
- If we apply 1000 iterations with time step size $\tau = 0.25$ and use the parameters $\lambda = 0.1$ and $\sigma = 0.8$, we obtain visually appealing results for both test images `trui.pgm` and `neworleans.pgm`.



`trui.pgm`

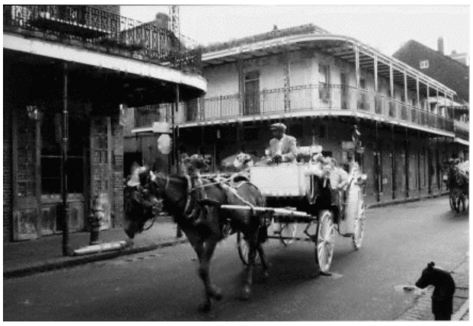


EED-inpainting



Since 1699, when French explorers
landed at the great bend of the
Mississippi River and celebrated
the first Mardi Gras in North America,
New Orleans has brewed a fascinating
melange of cultures. It was French,
then Spanish, then French again, then
sold to the United States. Through all
these years, and even into the 1900s,
others arrived from everywhere:
Acadians (Cajuns), Africans, indige-

`neworleans.pgm`



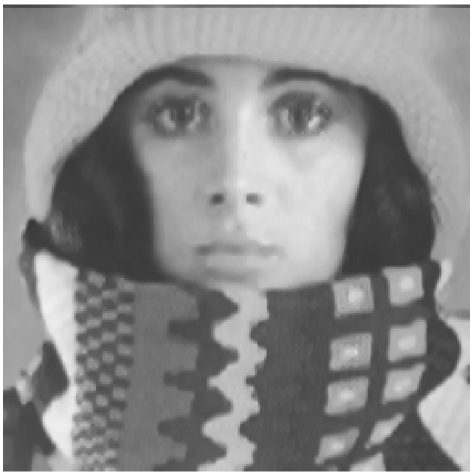
EED-inpainting

- In order to determine the quality of our inpainting results we had to supplement the file `error.c` with a routine for computing the average Euclidean error (AEE) and the average absolute error (AAE). The complete routine is given by

```
/* -------------------------------------------------------- */
void compute_errors

     (float **f,   /* computed image */
      float **t,   /* grount truth image */
      long  nx,    /* size in x direction */
      long  ny,    /* size in y direction */
      float *aae,  /* average absolute error */
      float *aee)  /* average Euclidean error */

     /* computes the AAE and the AEE error */

{
long i,j;     /* loop variables */
long n;       /* total number of pixels */

/* compute total number of pixels */
n=nx*ny;

/* initialise error variables */
*aae=0.0;
*aee=0.0;


/*compute AAE and AEE error */
for (i=1; i<=nx; i++)
  for (j=1; j<=ny; j++)
  {
    *aae += abs(f[i][j] - t[i][j]);
    *aee += (f[i][j] - t[i][j]) * (f[i][j] - t[i][j]);
  }

*aae = *aae / n;
*aee = sqrt(*aee / n);

 return;
}
/* -------------------------------------------------------- */
```

This allowed us to compare our results for $\lambda = 0.1$ and $\sigma = 0.8$ with the original image given by `trui_orig.pgm`.

original


EED-inpainting

The AAE for this result is given by 3.00 pixels, the AEE is 7.31 pixels. Please note that a further optimisation of the parameters my still allow an improvement of the results.