Image Processing and Computer Vision 2005/06
Example Solutions for Practical Assignments 3 (P3)

Problem 1.

With the subroutines for erosion and dilation one can implement the missing subroutines according to the lecture notes.

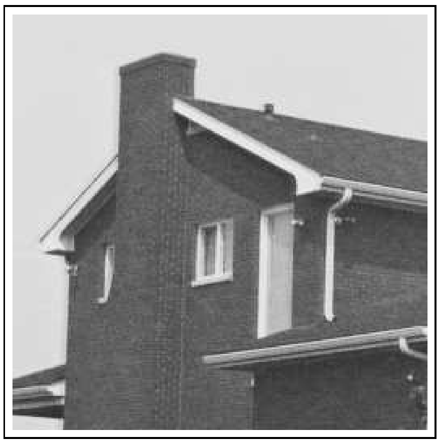
```
/*--------------------------------------------------------------------*/

void closing
 (long     nx,   /* image dimension in x direction */
  long     ny,   /* image dimension in y direction */
  long     m,    /* size of structuring element: (2m+1)*(2m+1) */
  float    **u)  /* input: original image; output: processed */

/*
 Closing with a square of size (2m + 1) * (2m + 1) as
 structuring element.
*/

{
dilation(nx, ny, m, u);
erosion(nx, ny, m, u);

return;
}

/*--------------------------------------------------------------------*/

void opening
 (long     nx,   /* image dimension in x direction */
  long     ny,   /* image dimension in y direction */
  long     m,    /* size of structuring element: (2m+1)*(2m+1) */
  float    **u)  /* input: original image; output: processed */

/*
 Opening with a square of size (2m + 1) * (2m + 1) as
 structuring element.
*/
```

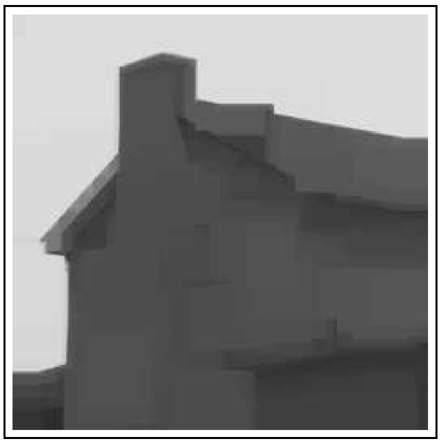
```
{
erosion(nx, ny, m, u);
dilation(nx, ny, m, u);

return;
}

/*--------------------------------------------------------------*/

void white_top_hat
 (long     nx,   /* image dimension in x direction */
  long     ny,   /* image dimension in y direction */
  long     m,    /* size of structuring element: (2m+1)*(2m+1) */
  float    **u)  /* input: original image; output: processed */

/*
 White top hat with a square of size (2m + 1) * (2m + 1) as
 structuring element.
*/

{
long    i, j;       /* loop variables */
float   **uo;       /* opening of input image */

alloc_matrix (&uo, nx+2, ny+2);

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

/* perform opening */
opening(nx, ny, m, uo);

/* subtract opening from u */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++)
         u[i][j] = u[i][j] - uo[i][j];

disalloc_matrix (uo, nx+2, ny+2);
return;
} /* white_top_hat */

/*--------------------------------------------------------------------*/

void black_top_hat
 (long     nx,   /* image dimension in x direction */
  long     ny,   /* image dimension in y direction */
  long     m,    /* size of structuring element: (2m+1)*(2m+1) */
  float    **u)  /* input: original image; output: processed */

/*
 Black top hat with a square of size (2m + 1) * (2m + 1) as
 structuring element.
*/

{
long    i, j;       /* loop variables */
float   **uc;       /* closing of input image */

alloc_matrix (&uc, nx+2, ny+2);

/* copy u info uc */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++)
        uc[i][j] = u[i][j];

/* perform closing */
closing(nx, ny, m, uc);

/* subtract u from closing */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++)
        u[i][j] = uc[i][j] - u[i][j];

disalloc_matrix (uc, nx+2, ny+2);
return;

} /* black_top_hat */

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

```
void selfdual_top_hat
 (long     nx,   /* image dimension in x direction */
  long     ny,   /* image dimension in y direction */
  long     m,    /* size of structuring element: (2m+1)*(2m+1) */
  float    **u)  /* input: original image; output: processed */

/*
 Black top hat with a square of size (2m + 1) * (2m + 1) as
 structuring element.
*/

{
long    i, j;       /* loop variables */
float   **uc;       /* copy of input image */

alloc_matrix (&uc, nx+2, ny+2);

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

closing(nx, ny, m, uc);
opening(nx, ny, m, u);

/* compute the difference between u and uc */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++)
         u[i][j] = uc[i][j] - u[i][j];

disalloc_matrix (uc, nx+2, ny+2);
return;

} /* selfdual_top_hat */

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

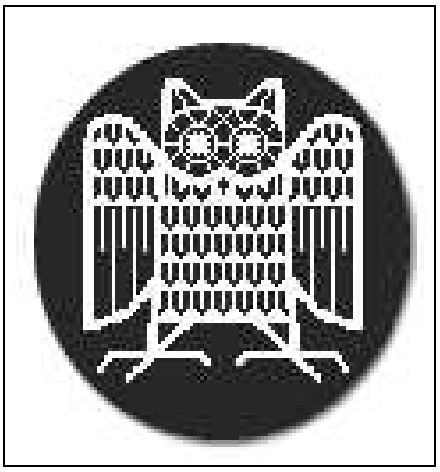
It is also possible to implement the selfdual top hat as the sum of black and the white top hat. The formula used here avoids to add and subtract the initial image f and thus needs less elementary operations.

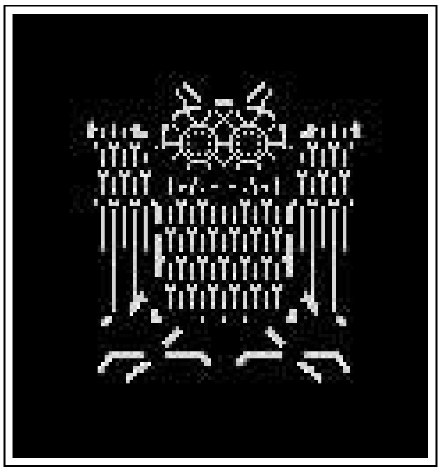
Now we show some examples how to apply this implementation on the problems given in the exercise sheet. Usually there are various ways to solve the problems. The parameters presented here should just give an idea how to proceed. For the removal of windows and doors it is possible to use an opening operation with a suitable structure element size.

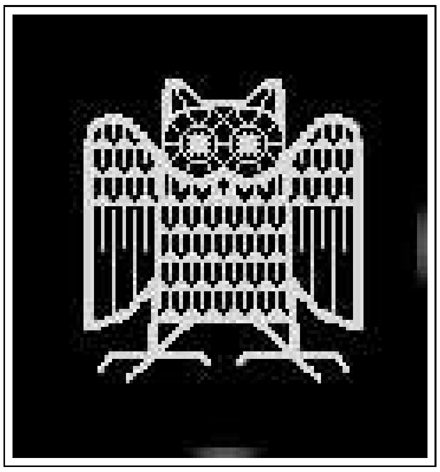

Original image

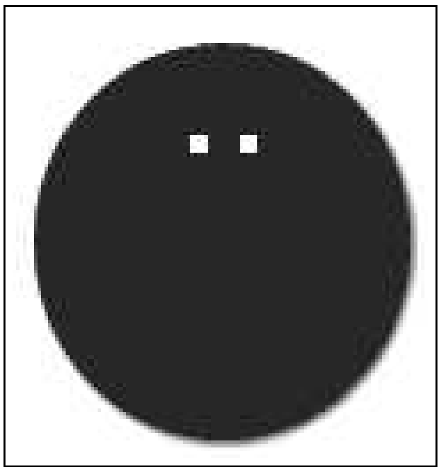

Opening ($m = 15$)

University owls are very shy at night – only a few people have ever seen one. We show different ways how one can imagine such a species:

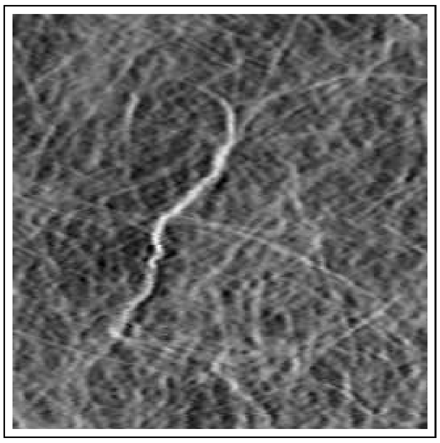

Original image

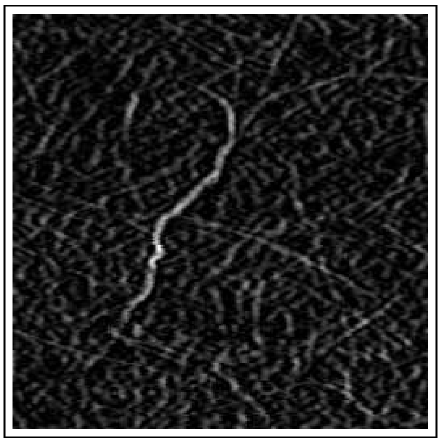

White top hat ($m = 1$)


White top hat ($m = 3$)

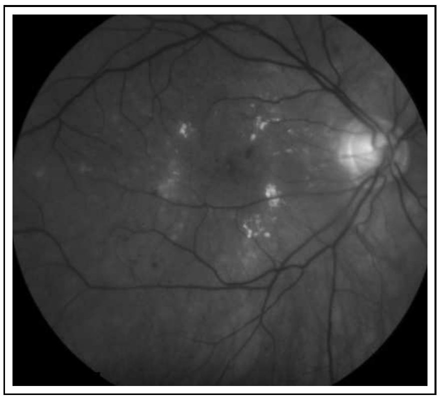

Opening ($m = 2$)

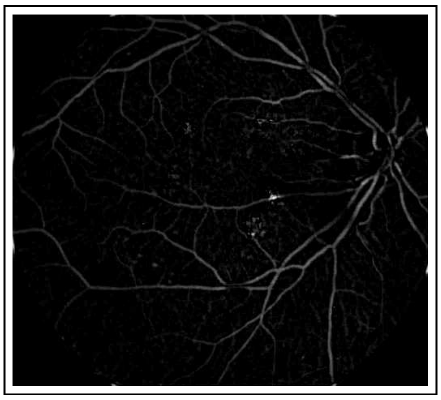
To separate small structures from the background some top hat is the method of choice. Here we have used a white top hat to emphasise the bright structures. For both the fabric image and the angiogram it is useful to normalise the results with xv.


Original image


White top hat ($m = 3$)
normalised with xv

For the angiogram we aim to visualise the dark structures, that means we are applying a black top hat.

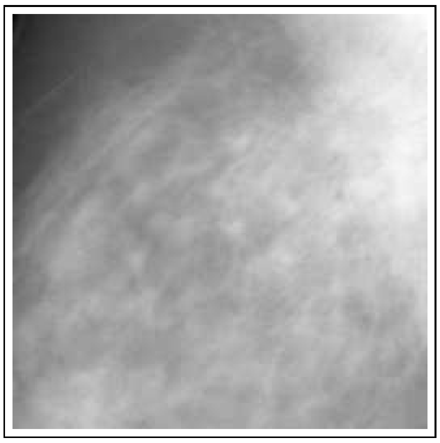

Original image

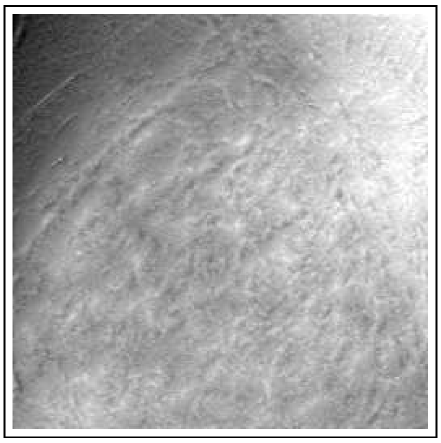

Black top hat ($m = 3$)
normalised with xv

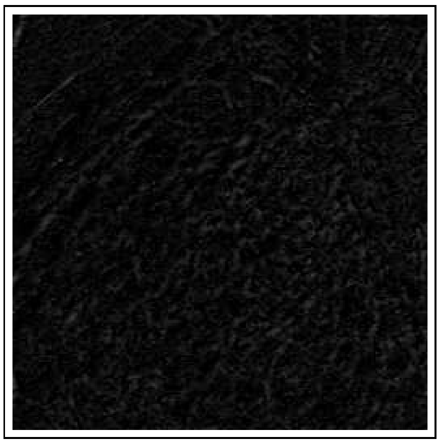
Problem 2 (Morphological Contrast Enhancement)

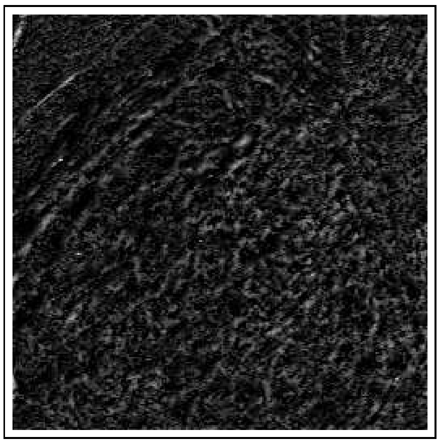
```
/*--------------------------------------------------------------*/

void contrast_enhancement
 (long     nx,      /* image dimension in x direction */
  long     ny,      /* image dimension in y direction */
  long     m,       /* size of structuring element: (2m+1)*(2m+1) */
  float    factor,  /* enhancement factor; 0 gives no enhancement */
  float    **u)     /* input: original image; output: processed */

/*
 Contrast enhancement using black and white top hats.
*/

{
long    i, j;         /* loop variables */
float   **uw, **ub;   /* white and black top hats */

alloc_matrix (&uw, nx+2, ny+2);
alloc_matrix (&ub, nx+2, ny+2);

/* copy u into uw and ub */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++) {
        uw[i][j] = u[i][j];
        ub[i][j] = u[i][j];
    }

/* compute top hats */
white_top_hat(nx, ny, m, uw);
black_top_hat(nx, ny, m, ub);

/* perform contrast enhancement */
for(i=1; i<=nx; i++)
    for(j=1; j<=ny; j++)
        u[i][j] = u[i][j] + factor * ( uw[i][j] - ub[i][j] );

disalloc_matrix (uw, nx+2, ny+2);
disalloc_matrix (ub, nx+2, ny+2);

} /* contrast_enhancement */

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

To visualise calcifications and star-shapes structures we proceed in several steps. First we use the subroutine `contrast_enhancement` implemented above. A white top hat then extracts white details of the image. With a final normalisation we can make small structures visible.


1: Original image


2: Contrast enhancement ($m = 2$, factor= 3)


3: White top hat of (**2**) ($m = 3$)


4: Normalised version of (**3**)