Image Processing and Computer Vision (IPCV)

Prof. J. Weickert
Mathematical Image Analysis Group
Winter term 13/14
Saarland University

Example Solutions for Homework Assignment 11 (H11)

Problem 1: (Segmentation Methods)

(a) For region growing, for a given threshold T we merge all points that are adjacent to the seed and differ less than T from their direct region neighbours. We start with the lowest possible threshold, $T = 1$ and increase it to the highest possible threshold, $T = 8$.

$$
\begin{aligned}
T = 1 \quad & \boldsymbol{f} = (0, 1, 6, 7, 5, 2, 1, 5, 4, 3, \underline{2}, 4, 7, 7, 2, 1)^\top \\
T = 2 \quad & \boldsymbol{f} = (0, 1, 6, 7, 5, 2, 1, \underline{5, 4, 3, 2}, 4, 7, 7, 2, 1)^\top \\
T = 3 \quad & \boldsymbol{f} = (0, 1, 6, 7, 5, 2, 1, \underline{5, 4, 3, 2, 4}, 7, 7, 2, 1)^\top \\
T = 4 \quad & \boldsymbol{f} = (0, 1, 6, 7, 5, 2, 1, \underline{5, 4, 3, 2, 4, 7, 7}, 2, 1)^\top \\
T = 5 \quad & \boldsymbol{f} = (0, 1, \underline{6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7}, 2, 1)^\top \\
T = 6 \quad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top \\
T = 7 \quad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top \\
T = 8 \quad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top
\end{aligned}
$$

Region growing is used to separate one single object from the background. Thus, the underlined region is the object that grew from the manually marked seed and the rest of the image forms one background region. One can either assign binary values to both regions or assign the region's respective grey value to each of its members.

(b) For region merging, we start with the lowest threshold $T = 1$ and merge all pixels, where the neighbours differ by less than 1. To this end we write down the absolute values of the differences between neighbouring pixels:

$$\boldsymbol{f} = (0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1)^\top$$

differences: 1 5 1 2 3 1 4 1 1 1 2 3 0 5 1

We mark the different segments by underlines:

$$T = 1 \qquad \boldsymbol{f} = (\underline{0}, \underline{1}, \underline{6}, \underline{7}, \underline{5}, \underline{2}, \underline{1}, \underline{5}, \underline{4}, \underline{3}, \underline{2}, \underline{4}, \underline{7, 7}, \underline{2}, \underline{1})^\top$$

Now we increase T by one and merge all pixels where the neighbours differ by less then 2:

$$T = 2 \qquad \boldsymbol{f} = (\underline{0, 1}, \underline{6, 7}, \underline{5}, \underline{2, 1}, \underline{5, 4, 3, 2}, \underline{4}, \underline{7, 7}, \underline{2, 1})^\top$$

Note that the segments given by $T = 1$ are a subset of the segments given by $T = 2$.

The whole stack of signals arising when we use region merging and increase the threshold T from 1 in steps of 1 to 8 looks as follows:

$$\begin{aligned}
T = 1 \qquad & \boldsymbol{f} = (\underline{0}, \underline{1}, \underline{6}, \underline{7}, \underline{5}, \underline{2}, \underline{1}, \underline{5}, \underline{4}, \underline{3}, \underline{2}, \underline{4}, \underline{7, 7}, \underline{2}, \underline{1})^\top \\
T = 2 \qquad & \boldsymbol{f} = (\underline{0, 1}, \underline{6, 7}, \underline{5}, \underline{2, 1}, \underline{5, 4, 3, 2}, \underline{4}, \underline{7, 7}, \underline{2, 1})^\top \\
T = 3 \qquad & \boldsymbol{f} = (\underline{0, 1}, \underline{6, 7, 5}, \underline{2, 1}, \underline{5, 4, 3, 2, 4}, \underline{7, 7}, \underline{2, 1})^\top \\
T = 4 \qquad & \boldsymbol{f} = (\underline{0, 1}, \underline{6, 7, 5, 2, 1}, \underline{5, 4, 3, 2, 4, 7, 7}, \underline{2, 1})^\top \\
T = 5 \qquad & \boldsymbol{f} = (\underline{0, 1}, \underline{6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7}, \underline{2, 1})^\top \\
T = 6 \qquad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top \\
T = 7 \qquad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top \\
T = 8 \qquad & \boldsymbol{f} = (\underline{0, 1, 6, 7, 5, 2, 1, 5, 4, 3, 2, 4, 7, 7, 2, 1})^\top
\end{aligned}$$

In practice, the values of the merged regions are often assigned the average grey value of the region.

While region growing and region merging are very similar methods, they differ in key aspects. Region growing needs manually defined seed points and uses these to separate one single object from the background.

Region merging however creates many regions which are merged for higher thresholds. Due to the rather distinct choice of the object region in this signal, region merging produces similar results as region growing with the difference that the background is segmented into small regions.

(c) To determine the two thresholds, we first have to find the 50% and 75% quantile. The given signal has 16 pixels, and the grey values are distributed as follows:

Grey value	0	1	2	3	4	5	6	7
Absolute frequency	1	3	3	1	2	2	1	3
Cumulative frequency	1	4	7	8	10	12	13	16

The 50% quantile is the grey value x such that 50% of all pixels have a smaller grey value than x (see Lecture 21, slide 7). Thus the 50% quantile is $T_1 := 4$. Analogously the 75% quantile is $T_2 := 6$.

NB: Sometimes, there is a "smaller or equal" instead of the "smaller" in the definition of the quantile. This would lead to the values $T_1 = 3$ and $T_1 = 5$ in our case. We proceed with the values $T_1 = 4$ and $T_2 = 6$ here because they are in accordance with the definition given in the lecture.

The first step of double thresholding selects all pixels with grey values larger than $T_2 = 6$. This leads to the following segmentation:

$$\boldsymbol{f} = (0, 1, 6, \underline{7}, 5, 2, 1, 5, 4, 3, 2, 4, \underline{7, 7}, 2, 1)^\top \ .$$

In the second step, all pixels in a direct neighbourhood with grey values larger than $T_1 = 4$. This gives the final result

$$\boldsymbol{f} = (0, 1, \underline{6, 7, 5}, 2, 1, 5, 4, 3, 2, 4, \underline{7, 7}, 2, 1)^\top \ .$$

For example, the 5 at position 8 is not added, since there is no seed point in the vicinity.

Problem 2: (Mumford-Shah Cartoon Model)

The Mumford Shah cartoon model uses the variational ansatz

$$E(K) = \int_{\Omega \setminus K} |u - f|^2 \, \mathrm{d}x\mathrm{d}y + \lambda \ell(K)$$

where $\ell(K)$ measures the length of K and $\lambda > 0$ is a scale parameter. We assume that $\Omega_1, \ldots, \Omega_n$ is a decomposition of Ω into disjoint regions with boundaries K. The segments are separated completely by K such that they can be treated independently for the minimisation of the energy. Thus the function u minimising the energy is constant in each segment and equal to the mean value of f in the segment:

$$u_m = \frac{1}{|\Omega_m|} \int_{\Omega_m} f \ dx \ dy \quad \text{for} \quad m = 1, \ldots, n \tag{1}$$

This can be seen by setting the derivative of the similarity term to zero. Let Ω_i, Ω_j denote two different segments. Merging the two segments Ω_i and Ω_j thus creates a new function v which is equal to u in all segments except in the new segment $\Omega_i \cup \Omega_j$ where it has the value

$$\tilde{u} := \frac{1}{|\Omega_i| + |\Omega_j|} \int_{\Omega_i \cup \Omega_j} f \, \mathrm{d}x\mathrm{d}y = \frac{|\Omega_i| u_i + |\Omega_j| u_j}{|\Omega_i| + |\Omega_j|} \tag{2}$$

Note that this value is a weighted average of the previous values. The weights are determined by the sizes of the regions Ω_i and Ω_j. Now, we want to show that merging the two regions Ω_i and Ω_j results in the following change of energy:

$$E(K \setminus \partial(\Omega_i, \Omega_j)) - E(K) = \frac{|\Omega_i| \cdot |\Omega_j|}{|\Omega_i| + |\Omega_j|} |u_i - u_j|^2 - \lambda \ell(\partial(\Omega_i, \Omega_j))$$

where $\partial(\Omega_i, \Omega_j)$ denotes the common boundary between Ω_i and Ω_j.

We write down the difference between the energy values before and after merging:

$$\begin{aligned} &E(K \setminus \partial(\Omega_i, \Omega_j)) - E(K) = \\ &\quad \int_{\Omega \setminus (K \setminus \partial(\Omega_i, \Omega_j))} |v - f|^2 \, \mathrm{d}x\mathrm{d}y + \lambda \ell(K \setminus \partial(\Omega_i, \Omega_j)) \\ &\quad - \int_{\Omega - K} |u - f|^2 \, \mathrm{d}x\mathrm{d}y - \lambda \ell(K). \end{aligned}$$

The length of the boundary is reduced by the length of the common boundary between Ω_i and Ω_j during the merging step. The difference between the length terms thus can be expressed as

$$\ell(K \setminus \partial(\Omega_i, \Omega_j)) - \ell(K) = -\ell(\partial(\Omega_i, \Omega_j)) \tag{3}$$

Now we take a closer look at the similarity term. With $M = \{1, \ldots, n\}$ we denote the index set of our regions. Considering the fact that u and v are constant in each region, we split up the integrals in sums over the several regions

$$\int_{\Omega \setminus K} |u - f|^2 \;\; = \;\; \sum_{m \in M} \int_{\Omega_m} |u_m - f|^2 \, \mathrm{d}x\mathrm{d}y$$

In the second case we remember that v equals u in all regions except $\Omega_i \cup \Omega_j$:

$$\begin{aligned} \int_{\Omega \setminus (K \setminus \partial(\Omega_i, \Omega_j))} |v - f|^2 \, \mathrm{d}x\mathrm{d}y \;\; = \;\; &\sum_{m \in M \setminus \{i,j\}} \int_{\Omega_m} |u_m - f|^2 \, \mathrm{d}x\mathrm{d}y \\ &+ \int_{\Omega_i \cup \Omega_j} |\tilde{u} - f|^2 \, \mathrm{d}x\mathrm{d}y \end{aligned}$$

The difference between the two similarity terms then can be written as

$$\begin{aligned}
&\int_{\Omega\setminus(K\setminus\partial(\Omega_i,\Omega_j))} |v-f|^2 \,\mathrm{d}x\mathrm{d}y - \int_{\Omega\setminus K} |u-f|^2 \,\mathrm{d}x\mathrm{d}y \\
&= \int_{\Omega_i\cup\Omega_j} |\tilde{u}-f|^2 \,\mathrm{d}x\mathrm{d}y - \int_{\Omega_i} |u_i-f|^2 \,\mathrm{d}x\mathrm{d}y - \int_{\Omega_j} |u_j-f|^2 \,\mathrm{d}x\mathrm{d}y \\
&= \int_{\Omega_i\cup\Omega_j} \left(\tilde{u}^2 - 2\tilde{u}f + f^2\right) \,\mathrm{d}x\mathrm{d}y - \int_{\Omega_i} \left(u_i^2 - 2u_i f + f^2\right) \,\mathrm{d}x\mathrm{d}y \\
&\quad - \int_{\Omega_j} \left(u_j^2 - 2u_j f + f^2\right) \,\mathrm{d}x\mathrm{d}y \\
&= \int_{\Omega_i\cup\Omega_j} \left(\tilde{u}^2 - 2\tilde{u}f\right) \,\mathrm{d}x\mathrm{d}y - \int_{\Omega_i} \left(u_i^2 - 2u_i f\right) \,\mathrm{d}x\mathrm{d}y - \int_{\Omega_j} \left(u_j^2 - 2u_j f\right) \,\mathrm{d}x\mathrm{d}y \\
&= \left(|\Omega_i| + |\Omega_j|\right)\tilde{u}^2 - 2\tilde{u}\int_{\Omega_i\cup\Omega_j} f \,\mathrm{d}x\mathrm{d}y - |\Omega_i|u_i^2 + 2u_i\int_{\Omega_i} f \,\mathrm{d}x\mathrm{d}y \\
&\quad -|\Omega_j|u_j^2 + 2u_j\int_{\Omega_j} f \,\mathrm{d}x\mathrm{d}y
\end{aligned}$$

We use the definition of $\tilde{u}$, u_i and u_j as mean values in (1) and (2) to transform this expression to

$$\left(|\Omega_i| + |\Omega_j|\right)\tilde{u}^2 - 2\left(|\Omega_i| + |\Omega_j|\right)\tilde{u}^2 - |\Omega_i|u_i^2 + 2|\Omega_i|u_i^2 - |\Omega_j|u_j^2 + 2|\Omega_j|u_j^2$$

which can be immediately simplified to

$$|\Omega_i|u_i^2 + |\Omega_j|u_j^2 - \left(|\Omega_i| + |\Omega_j|\right)\tilde{u}^2 \tag{4}$$

Using the relation between $\tilde{u}$, u_i and u_j given in (2), we can finally write this with a common denominator as

$$\begin{aligned}
&|\Omega_i|u_i^2 + |\Omega_j|u_j^2 - \left(|\Omega_i| + |\Omega_j|\right)\tilde{u}^2 \\
&= \frac{1}{|\Omega_i| + |\Omega_j|}\left(\left(|\Omega_i| + |\Omega_j|\right)\left(u_i^2|\Omega_i| + u_j^2|\Omega_j|\right) - \left(|\Omega_i|u_i + |\Omega_j|u_j\right)^2\right) \\
&= \frac{1}{|\Omega_i| + |\Omega_j|}\left(|\Omega_i|\cdot|\Omega_j|\left(u_i^2 - 2u_iu_j + u_j^2\right)\right) \\
&= \frac{|\Omega_i|\cdot|\Omega_j|}{|\Omega_i| + |\Omega_j|}|u_i - u_j|^2
\end{aligned} \tag{5}$$

Together the equations (5) and (3) show the formula we wanted to prove:

$$E(K\setminus\partial(\Omega_i,\Omega_j)) - E(K) = \frac{|\Omega_i|\cdot|\Omega_j|}{|\Omega_i| + |\Omega_j|}|u_i - u_j|^2 - \lambda\ell(\partial(\Omega_i,\Omega_j))$$

Problem 3 (Toboggan Watershed Segmentation)

(a) In the framework of the programming exercise, region merging is implemented with stack structures. Each point of the image domain is visited exactly once in the process; the matrix *visited* is used as a lookup table for already visited points. The algorithm iterates over all image points, column by column. If a point $p = (i, j)$ is found that was not visited, yet, it is used as the seed for a new segment. p is added to the stack *segment* and all of its direct neighbours n_k ($k \in \{1, \ldots, 8\}$) that fulfil the merging criterion $n_k < T$ and were not yet visited, are added to the stack *to_be_visited*. Successively, all of the points from *to_be_visited* are inserted into the *segment* stack, again adding direct neighbours to *to_be_visited* with the same rules as described above.

This procedure is repeated until all points that belong to the segment are found, i.e. *to_be_visited* is empty. All points from the segment stack are then assigned the average grey value of the segment points. After a segment is computed, the algorithm continues the column-wise iteration over the image, discarding visited points that already belong to a segment and starting new region merging processes if another seed is found.

```
/*-------------------------------------------------------------*/
/* start region merging */
for (i=1; i<=nx; i++)
  for (j=1; j<=ny; j++)
  {
    /* if pixel was not visited yet start looking
        for a new segment */
    if (visited[i][j] != 1.0f)
    {
      seg_mean = 0.0f;
      sum_pixel = 0;

      /* add pixel to the set of pixels which have
          still to be visited for the currently
          considered segment */
      push_Stack(to_be_visited,new_intPair(i,j));

      while (!empty_Stack(to_be_visited))
      {
```

```
/* get a pixel which has still to be visited */
x = ((intPair*)front_Stack(to_be_visited))->x;
y = ((intPair*)front_Stack(to_be_visited))->y;

/* remove it from "to_be_visited" stack */
pop_Stack(to_be_visited);

if (visited[x][y] != 1.0f)
{
  /* mark pixel as visited
      and add it to the current segment */
  visited[x][y] = 1.0f;
  push_Stack(segment,new_intPair(x,y));

  sum_pixel++;
  seg_mean += f[x][y];

  /* if neighbour has not been visited yet
      and belongs to the same segment add
      to "to_be_visited" stack */

  // BEGIN SUPPLEMENTED CODE
  if (!(visited[x+1][y]==1.0f) &&
        (fabs(f[x][y]-f[x+1][y]) < T))
      push_Stack(to_be_visited,new_intPair(x+1,y));
  if (!(visited[x+1][y+1]==1.0f) &&
        (fabs(f[x][y]-f[x+1][y+1]) < T))
      push_Stack(to_be_visited,new_intPair(x+1,y+1));
  if (!(visited[x+1][y-1]==1.0f) &&
        (fabs(f[x][y]-f[x+1][y-1]) < T))
      push_Stack(to_be_visited,new_intPair(x+1,y-1));
  if (!(visited[x-1][y]==1.0f) &&
        (fabs(f[x][y]-f[x-1][y]) < T))
      push_Stack(to_be_visited,new_intPair(x-1,y));
  if (!(visited[x-1][y-1]==1.0f) &&
        (fabs(f[x][y]-f[x-1][y-1]) < T))
      push_Stack(to_be_visited,new_intPair(x-1,y-1));
  if (!(visited[x-1][y+1]==1.0f) &&
        (fabs(f[x][y]-f[x-1][y+1]) < T))
      push_Stack(to_be_visited,new_intPair(x-1,y+1));
  if (!(visited[x][y+1]==1.0f) &&
        (fabs(f[x][y]-f[x][y+1]) < T))
      push_Stack(to_be_visited,new_intPair(x,y+1));
```

```
            if (!(visited[x][y-1]==1.0f) &&
                  (fabs(f[x][y]-f[x][y-1]) < T))
                push_Stack(to_be_visited,new_intPair(x,y-1));
            // END SUPPLEMENTED CODE
          }
        }

        /* compute mean value of segment */
        seg_mean /= (float)sum_pixel;

        /* set value of segment pixels to mean value of segment */
        while (!empty_Stack(segment))
        {
            x = ((intPair*)front_Stack(segment))->x;
            y = ((intPair*)front_Stack(segment))->y;
            pop_Stack(segment);

            u[x][y] = seg_mean;
        }
      }
    }
/*-------------------------------------------------------------*/
```

Applying region merging to the two test pictures house.pgm and trui.pgm yields results that highly depend on the threshold T. A higher threshold produces larger segments. For $T = 1$, the image is always identical to the original, since only points that have the same value are merged. $T = 256$ always produces a single-colour image with the original's average grey value. Note that the results of region merging alone are not satisfactory, since even for thresholds that create segments that are larger than desired, other parts of the image will contain many very small regions that are barely bigger than one image point and thus almost identical to the original picture (see clusters of white segment boundaries). A good example is trui.pgm with $T = 5$. While many regions of the scarf and the face, that intuitively should be separated, are merged for this threshold, the left eye only underwent a very fine segmentation.

Original image house.pgm

Original image trui.pgm

$T = 5$

$T = 2$

$T = 10$

$T = 5$

Segment boundaries are indicated by white pixels

(b) As in (a), every point in the image is visited only once in the Toboggan watershed implementation. Again, the algorithm iterates over all points of the image, taking points that were not visited yet as starting points for the downhill path of the Toboggan. Beginning with the the starting point, successively the neighbour of the currently examined image point with the minimal gradient magnitude is chosen as the next point on the path, until a minimum is found. Each newly discovered path point is added to the stack s. After the path was completed, all the path points stored in s are assigned with the minimum.

Before the watershed algorithm above is applied to the image, it is presmoothed with a Gaussian of standard deviation σ.

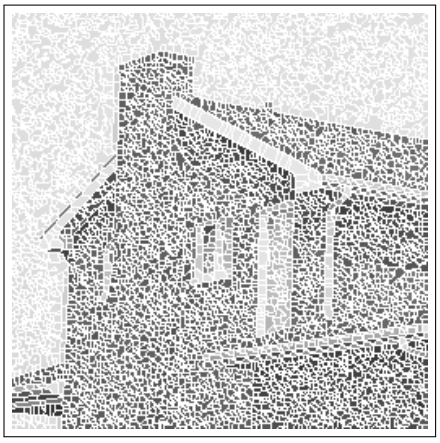
```
/* ---- perform watershed segmentation ---- */

int x, y; /* coordinates giving the current position of a "water drop" */
float color; /* colour found at minimum */
float min; /* auxillary variable giving the minimum in the
              8-neighbourhood of pixel (x,y) */
int next_x; /* x-coordinate of the minimum in the 8-neighbourhood */
int next_y; /* y-coordinate of the minimum in the 8-neighbourhood */

for (i=1; i<=nx; i++)
    for (j=1; j<=ny; j++)
    {
        /* if pixel was not visited yet */
        if (visited[i][j] != 1.0f)
        {

            /* put pixel on stack and mark as visited */
            x = i, y = j;
            push_Stack(s,new_intPair(x,y));
            visited[i][j] = 1.0f;

            /* while minimum not reached follow the water drop on its path
               down to minimum */
            int minimum_reached = 0;
            while (!minimum_reached)
            {

                if (   gradmagn[x][y]>gradmagn[x+1][y  ]
                    || gradmagn[x][y]>gradmagn[x-1][y  ]
                    || gradmagn[x][y]>gradmagn[x  ][y+1]
                    || gradmagn[x][y]>gradmagn[x  ][y-1]
```

```
    || gradmagn[x][y]>gradmagn[x+1][y+1]
    || gradmagn[x][y]>gradmagn[x+1][y-1]
    || gradmagn[x][y]>gradmagn[x-1][y+1]
    || gradmagn[x][y]>gradmagn[x-1][y-1])
{
    /* weight for diagonal distance */
    const float sqrt2 = 1.0f/sqrtf(2.0f);

    /* initialise with one neighbour */
    min = (gradmagn[x+1][y+1]-gradmagn[x][y])*sqrt2;
    next_x = x+1;
    next_y = y+1;

    /* check for steepest descent
       and update next_x, next_y and min */
    if ((gradmagn[x-1][y-1]-gradmagn[x][y])*sqrt2 < min)
    {
      next_x = x-1;
      next_y = y-1;
      min = (gradmagn[x-1][y-1]-gradmagn[x][y])*sqrt2;
    }
    if ((gradmagn[x-1][y+1]-gradmagn[x][y])*sqrt2 < min)
    {
      next_x = x-1;
      next_y = y+1;
      min = (gradmagn[x-1][y+1]-gradmagn[x][y])*sqrt2;
    }
    if ((gradmagn[x+1][y-1]-gradmagn[x][y])*sqrt2 < min)
    {
      next_x = x+1;
      next_y = y-1;
      min = (gradmagn[x+1][y-1]-gradmagn[x][y])*sqrt2;
    }
    if (gradmagn[x+1][y]-gradmagn[x][y] < min)
    {
      next_x = x+1;
      next_y = y;
      min = gradmagn[x+1][y]-gradmagn[x][y];
    }
    if (gradmagn[x][y+1]-gradmagn[x][y] < min)
    {
      next_x = x;
      next_y = y+1;
```

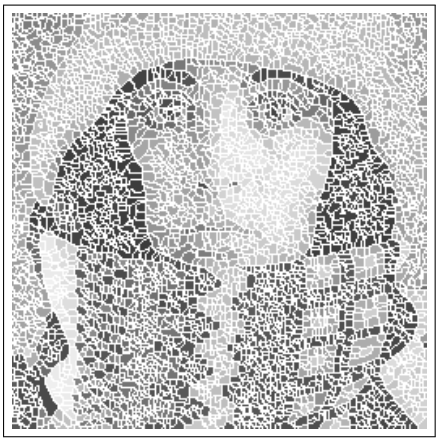
```
            min = gradmagn[x][y+1]-gradmagn[x][y];
          }
          if (gradmagn[x-1][y]-gradmagn[x][y] < min)
          {
            next_x = x-1;
            next_y = y;
            min = gradmagn[x-1][y]-gradmagn[x][y];
          }
          if (gradmagn[x][y-1]-gradmagn[x][y] < min)
          {
            next_x = x;
            next_y = y-1;
            min = gradmagn[x][y-1]-gradmagn[x][y];
          }

          /* set new current coordinates of waterdrop */
          x = next_x;
          y = next_y;
      }

      if (visited[x][y] != 1.0f)
      {     /* if pixel was not visited yet, mark pixel as visited
               and put it on stack */
          push_Stack(s,new_intPair(x,y));
          visited[x][y] = 1.0f;
      }
      else  /* if pixel was already visited stop since we know then
               the colour of the minimum */
          minimum_reached = 1;
  }

  /* get colour of minimum */
  color = f[x][y];

  /* mark all pixels on stack using the colour of the minimum */
  while (!(empty_Stack(s)))
  {
      x = ((intPair*)front_Stack(s))->x;
      y = ((intPair*)front_Stack(s))->y;
      pop_Stack(s);

      f[x][y] = color;
  }
```

```
        }
    }
```

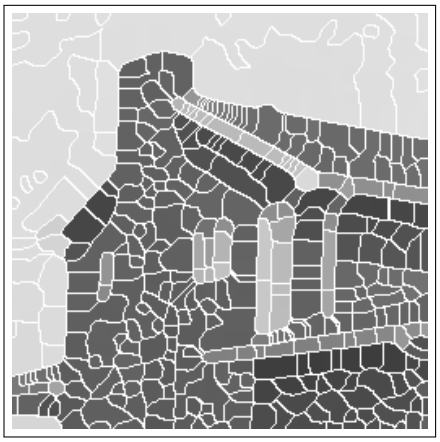
The presmoothing can be used to modify the size of the segments by removing and reallocating the minima of the image. While the results are better than in (b), they are still not optimal. If we do not presmooth ($\sigma = 0$) we get a highly oversegmented image. Even for a small amount of presmoothing, there are still too much regions. For higher presmoothing, the regions become indeed larger, but their boundaries are then dislocated. Additionally, higher amounts of presmoothing do not necessarily guarantee consistently larger segments, as can be seen by comparing house.pgm for $\sigma = 3.5$ and $\sigma = 10$.



$\sigma = 0$



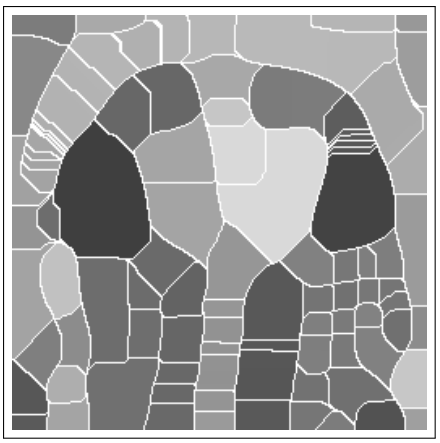
$\sigma = 0$



$\sigma = 3.5$

$\sigma = 3.5$

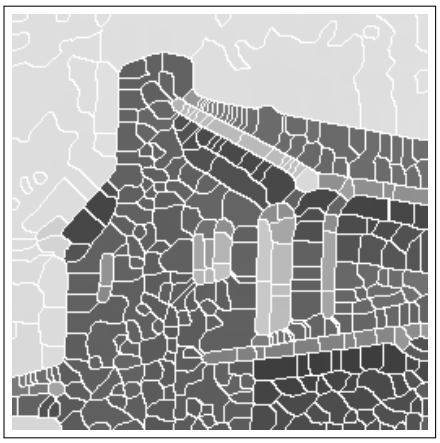
$\sigma = 10$



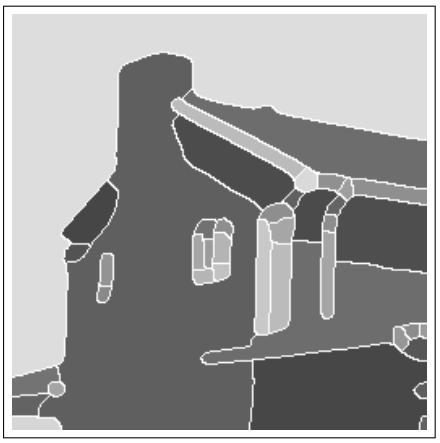
$\sigma = 10$

Segment boundaries are indicated by white pixels

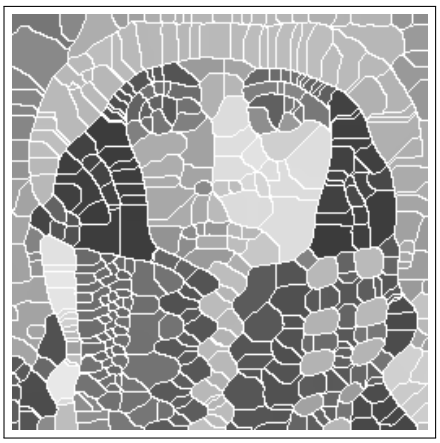
(c) As a remedy for the shortcomings of the region merging and watershed methods, both approaches can be combined. First, the Toboggan watershed algorithm is used on the image with a reasonable amount of presmoothing. Many of the resulting adjacent regions that should be merged for a higher quality segmentation have a similar greyvalue. Therefore, region merging can be applied on this preliminary, fine segmentation to create a coarser one with well located boundaries.



$\sigma = 3.5$



$\sigma = 3.5,\ T = 10$



$\sigma = 3.5$

$\sigma = 3.5,\ T = 10$

Segment boundaries are indicated by white pixels