Yen Duyen Amy Le

1001827177

CSE 4360

Homework 3

Implement Edge Detection Using Prewit Templates

I made a function for each edge type. In each function is a nested for loop that slides the appropriate template over all the pixels in the image (except for the pixels at the outer boundary), applying convolution to each pixel (since Prewit Templates only consists of -1, 0, and 1, and my brain was not fully working, I just either subtracted from or added to the total instead of multiplying then adding to the total to compute the cross-correlation values). I decided to make all cross-correlation values positive such that edges are generally shown in only one color (instead of both very dark and very light).

While computing the cross-correlation values, I kept track of the minimum and maximum cross-correlation values. I did this such that I can use the formula for slope of a linear function and the equation for the point-slope form of a linear function to normalize the cross-correlation values to the rand [0, 255].

I updated the size of the processed image to be the original image minus 2 for each dimension (because I didn't compute the cross-correlation values at the pixels at the outer boundary of the image).

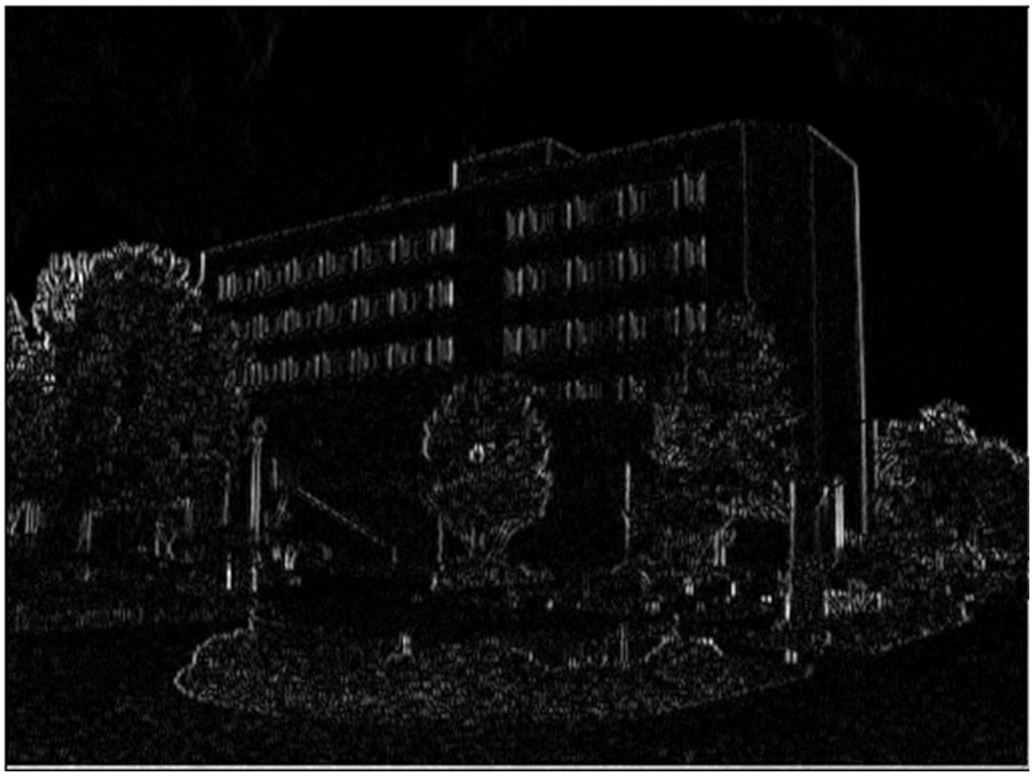

Figure 1. Vertical edge detection applied to nedderman.lpr

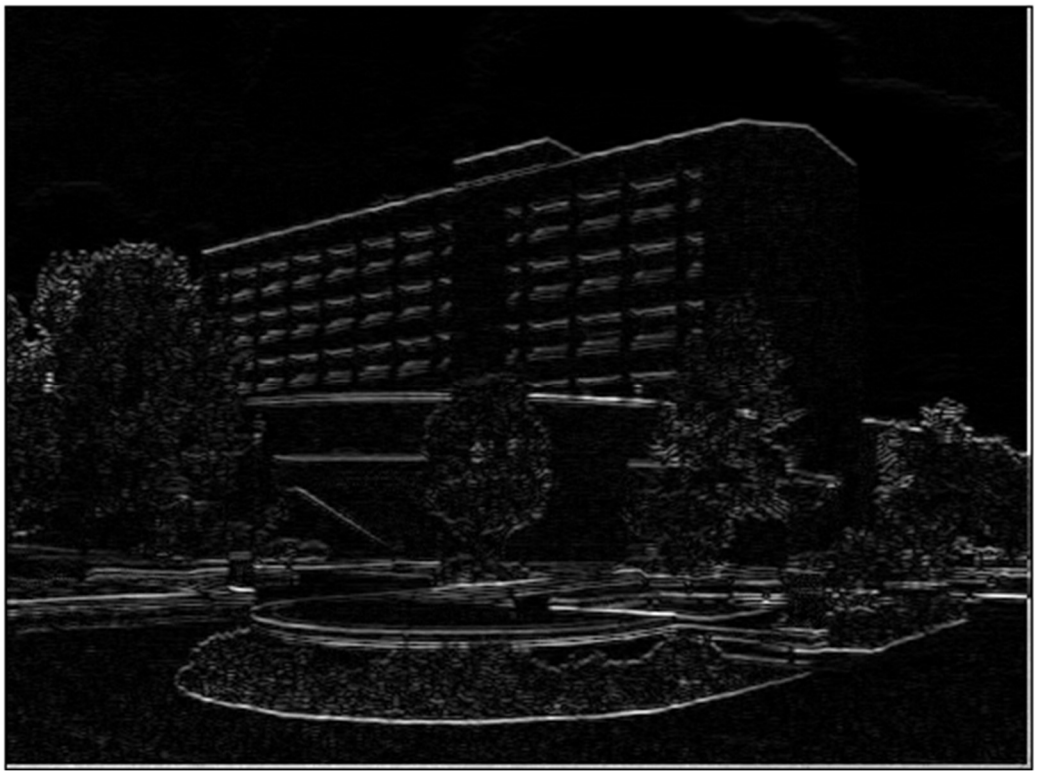

Figure 2. Horizontal edge detection applied to nedderman.lpr

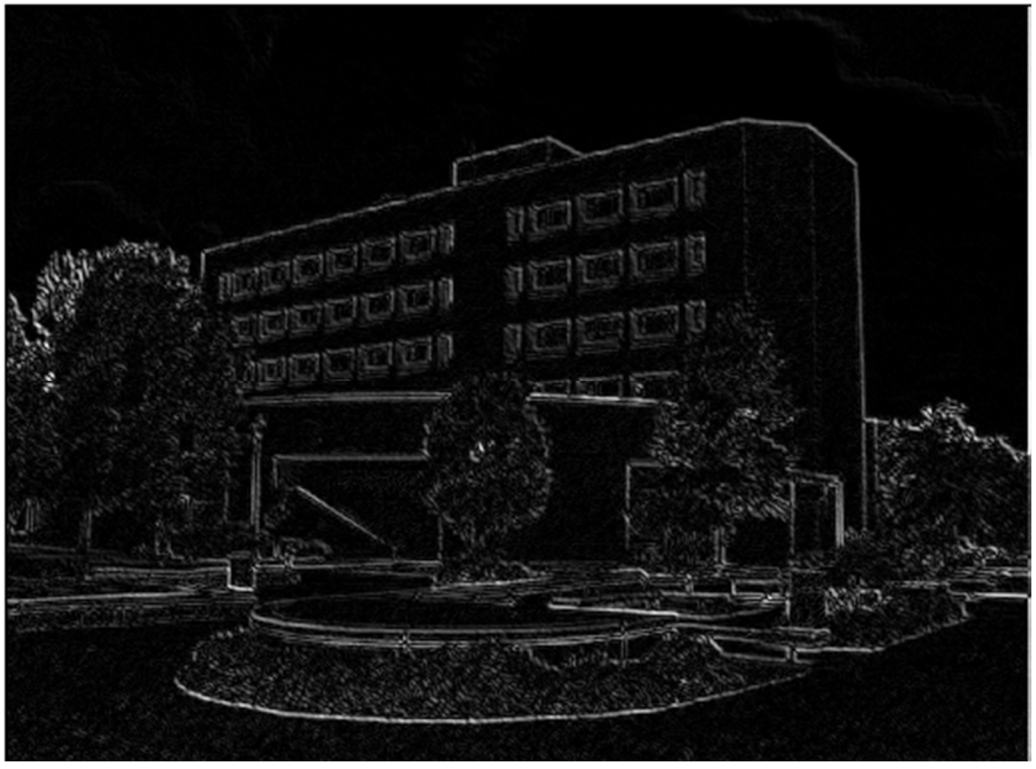

Figure 3. Major diagonal edge detection applied to nedderman.lpr

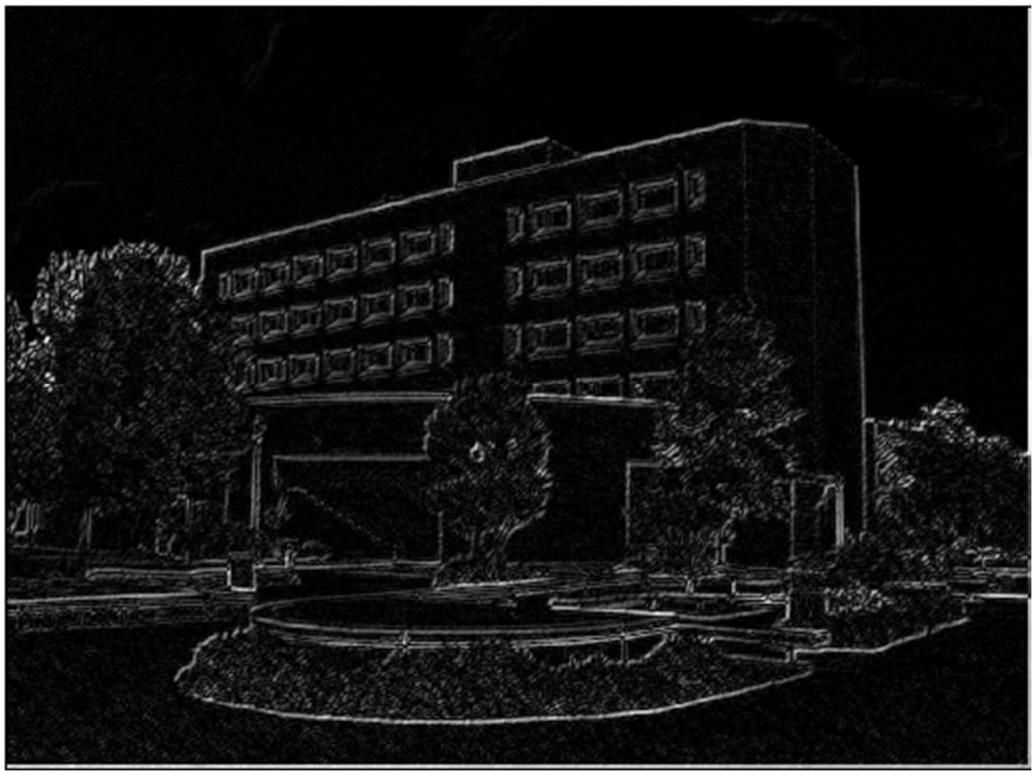

Figure 4. Minor diagonal edge detection applied to nedderman.lpr

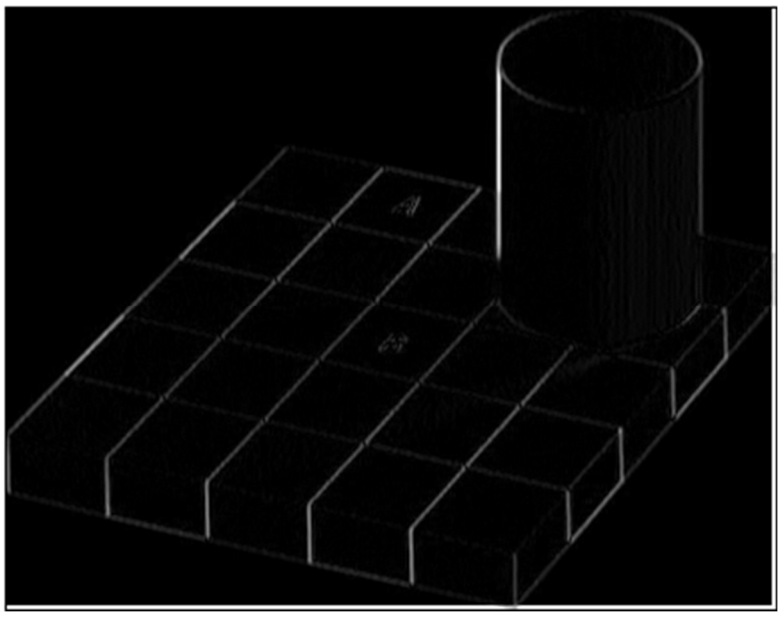

Figure 5. Vertical edge detection applied to chess.lpr

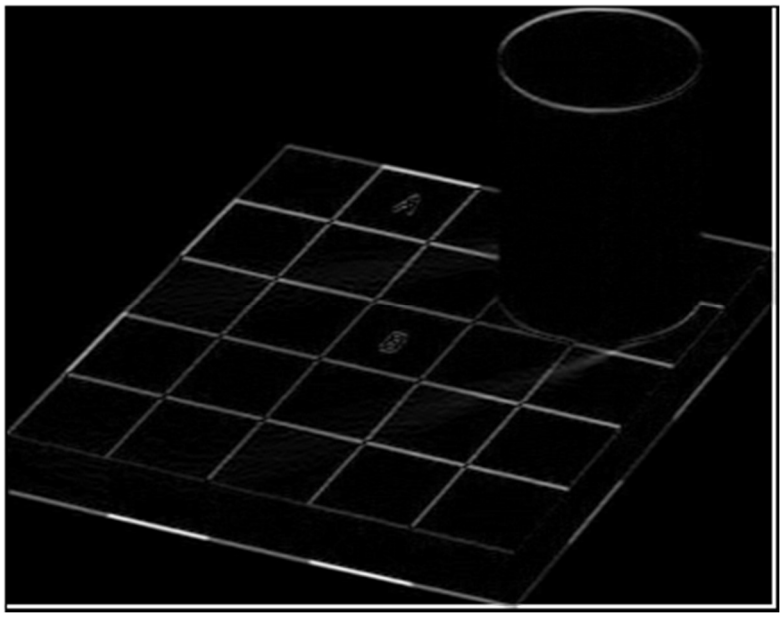

Figure 6. Horizontal edge detection applied to chess.lpr

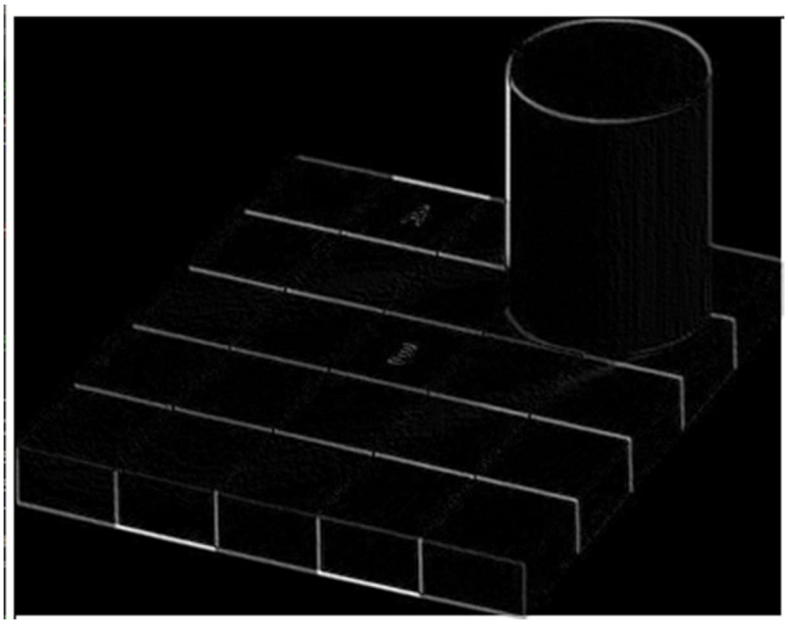

Figure 7. Major diagonal edge detection applied to chess.lpr

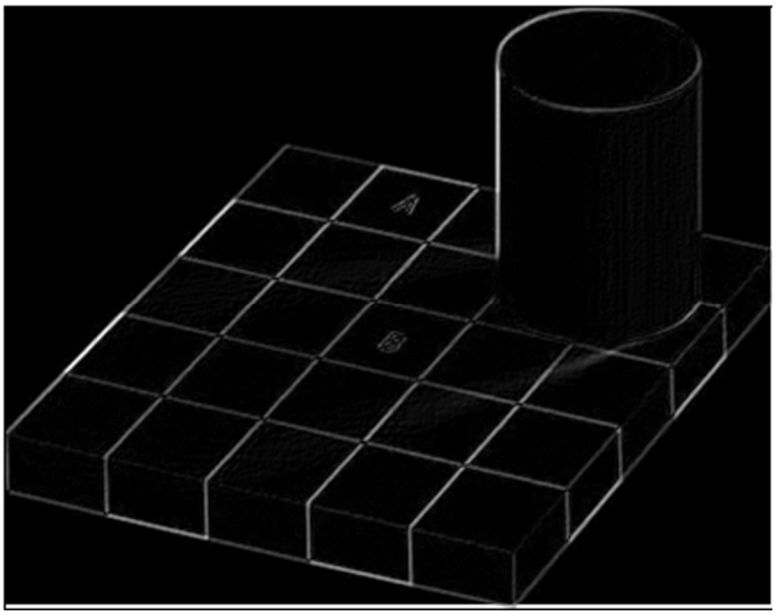

Figure 8. Minor diagonal edge detection applied to chess.lpr

Implement Template Matching Using Normalized Convolution

First, I found the average of the image. I then used the average to compute the standard deviation of the image. Before making calculations with the template, I forced the user-drawn ROI to have an odd height and width to apply convolution easier. I also found the average and the standard deviation of the template (aka user-drawn ROI) like I did for the image.

These averages and standard deviations were used to perform global normalized template matching. For each pixel in the image that is not at the boundaries (which depends on the size of the ROI), I calculated cross-correlation between the image region surrounding the pixel (the image region being the size of the user-drawn ROI) and the template region. Before computing each product, I made sure to subtract the averages from the image and the template to normalize for brightness. I also divided each product by both the standard deviation of the image and the standard deviation of the template to normalize for contrast.

While computing the cross-correlation values, I kept track of the minimum and maximum cross-correlation values. After completing the convolution, I normalized cross-correlation results to be in the range [0, 255] by applying the minimum and maximum values to the slope formula and the point-slope form equation.

Finally, I updated the size of the processed image to be the original image to be dependent on the size of the user-drawn ROI (since cross-correlation values were not able to be computed at the boundary of the image).

Figure 9. ROI drawn around the window in the top-right corner in nedderman.lpr

Figure 10. Result of template matching based on the ROI drawn in Figure 9

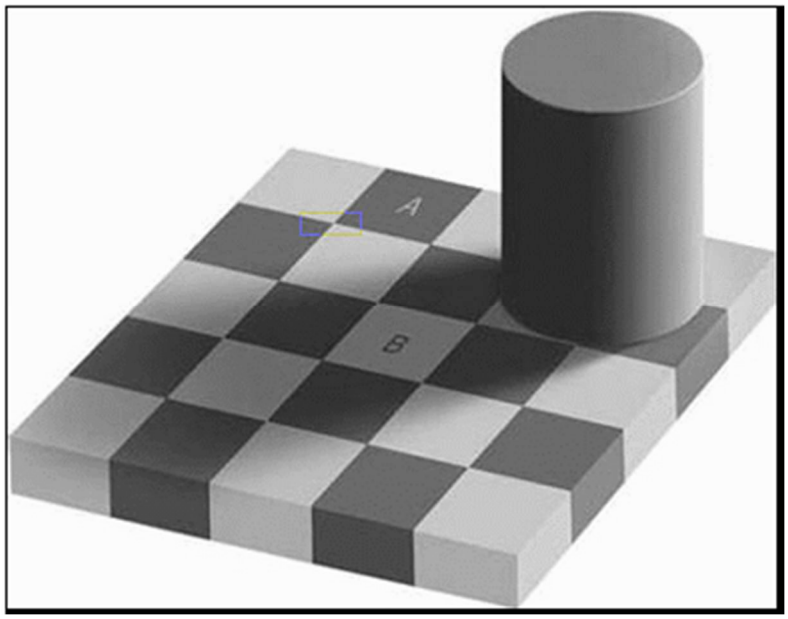

Figure 11. ROI drawn around the center of four squares in chess.lpr

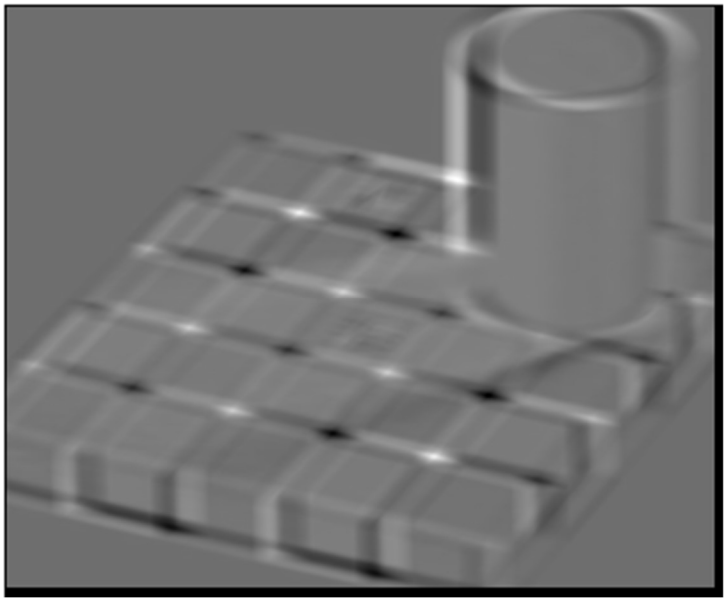

Figure 12. Result of template matching based on ROI drawn in Figure 11

Implement Segmentation Using Blob Coloring

The first time that I iterated through every single pixel of the image, it was to perform a “color pass”. I iterated from the top-left corner to the bottom-right corner. For each pixel, I either assigned it the region number of its top neighbor (if the intensity at the pixel fell was close enough to the intensity of the top neighbor pixel), its left neighbor (if the intensity at the pixel fell was close enough to the intensity of the left neighbor pixel), or a brand new region number. If the intensity at the pixel fell close

enough to both neighbors, but both neighbors had different region numbers, then I noted that the bigger region number can be mapped to the smaller region number in the future.

The second time that I iterated through every single pixel of the image, it was to perform a “dictionary pass”. I remapped regions that I previously noted during the first pass to the smallest region number possible.

Then, I generated an array of unique region numbers from the region numbers assigned during the second pass. For each region number, I normalized it to fall in the range [0, 255] such that I can use it as an index to pair unique regions to their own unique intensity value.

Finally, each pixel in the processed image was assigned the unique intensity value that was paired with the region number corresponding to that pixel that was assigned during the second pass.

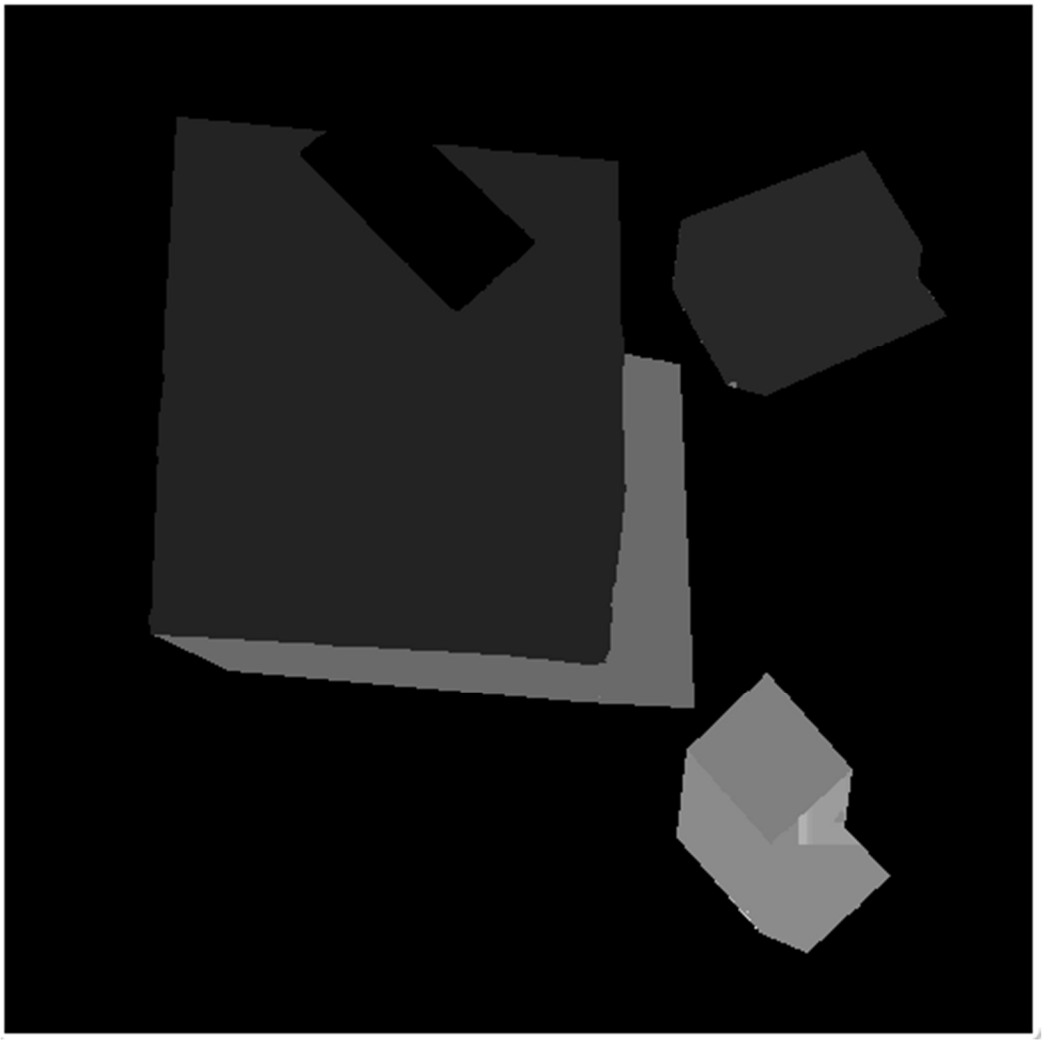

Figure 13. Result of blob coloring on blocks.lpr