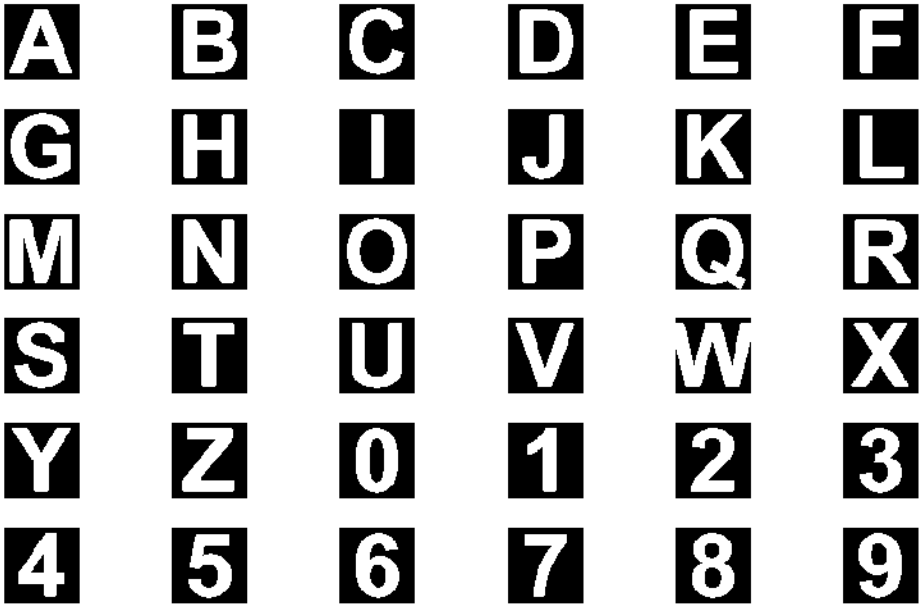
EE368/CS232 Digital Image Processing
Spring 2014-2015

Homework #3
Solutions

1. License Plate Recognition

Part A:
Below, we show the binarized versions of the 36 template images.



Below, we show the binarized version of the clean and noisy license plate images.

Part B:
To recognize the license plate characters by erosion, we

(i) Erode each template image by a 3x3 square structuring element to obtain a slightly smaller shape that covers the interior of each character
(ii) Erode the license plate image using the eroded template image as a structuring element
(iii) Dilate the eroded license plate image by the template image for visualization purposes

For every template image that generated a non-zero erosion result, we show the detected character instances below (a dimmed version of the original image is blended together with the dilation result for visualization purposes). The characters B, E, H, I, L, T, and 8 are correctly detected, but there are also false positive detections in the characters F, I, L, and 3.

Part C:
To recognize the license plate characters with a hit-miss filter, we

(i) Use the same structuring element as in Part A for the foreground

(ii) Define dilate(template, 5x5 square) – dilate(template, 3x3 square) as the structuring element for the background

(iv) Dilate the hit-miss result by the template image for visualization purposes

For every template image that generated a non-zero hit-miss result, we show the detected character instances below. Now, all the correct characters are successfully detected, and no false positives appear.

Part D:
For some chosen thresholds, the hit-miss filter can still correctly detect the letters in the noisy binarized image. But for other thresholds, some of the characters are not are detected, because the noise specks prevent the foreground and background matching conditions in the hit-miss filter from being fulfilled exactly.

Part E:
For the foreground and the background rank filters, we choose ranks of $p_1 = 15$ and $p_2 = 12$, respectively; other pairs of ranks might also give a correct result. Below, we show the detection results, where it can be observed that all of the correct characters are detected and false positives are avoided. The rank filter enables a partial match against the foreground/background templates of a character to result in a positive detection, thereby providing robustness against the noise in the license plate image while at the same time still being selective about which character is detected.

MATLAB Code:

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: License Plate Recognition
% Script by David Chen, Huizhong Chen

clc; clear all;

% Load and binarize templates
letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
templates = cell(1, length(letters));
threshold = 0.8;
figure(1); clf; set(gcf, 'Color', 'w');
for nLetter = 1:length(letters)
    img = im2double(imread(sprintf('Templates/%s.png', letters(nLetter))));
```

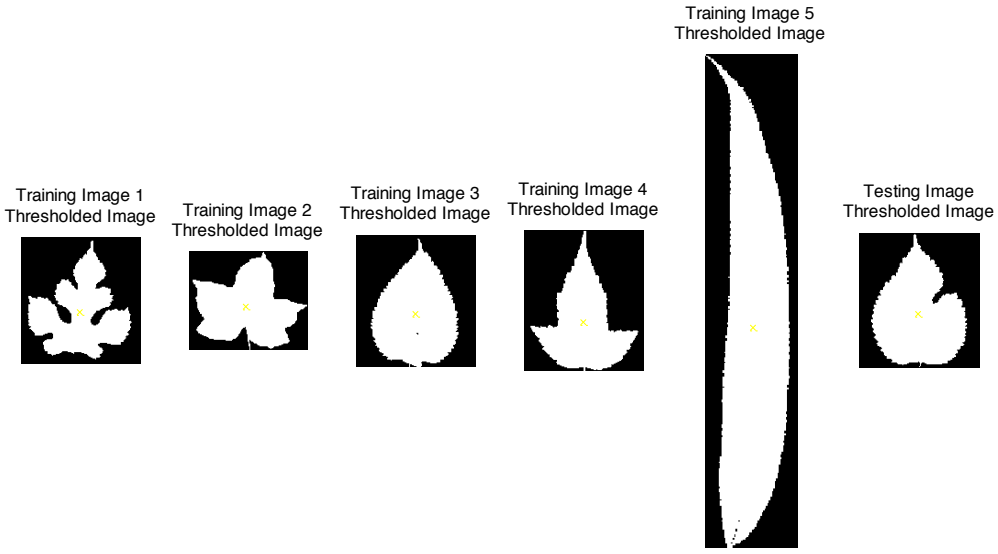
```
    imgGray = rgb2gray(img);
    templates{nLetter} = double(imgGray < threshold);
    subplot(6,6,nLetter);
    imshow(templates{nLetter});
end % nLetter

% Load and binarize test image
imgTest = im2double(imread('hw3_license_plate.png'));
imgTestGray = rgb2gray(imgTest);
[height,width] = size(imgTestGray);
imgTestBin = double(imgTestGray < threshold);
figure(2); clf; set(gcf, 'Color', 'w');
imshow(imgTestBin);

% Detect by erosion in crisp image
alpha = 0.4;
detectByErosion = 0;
if detectByErosion
    for nLetter = 1:length(letters)
        SE = imerode(templates{nLetter}, ones(3,3));
        imgTestErode = imerode(imgTestBin, SE);
        rows = 50:height-50; cols = 50:width-50;
        if sum(sum(imgTestErode(rows,cols))) > 0
            fprintf('Detected clean, erosion: %s\n', letters(nLetter));
            imgTestErodeDilate = imdilate(imgTestErode, SE);
            figure(3); clf; set(gcf, 'Color', 'w');
            imshow(max(imgTestErodeDilate, alpha*imgTestBin));
            pause
        end
    end % nLetter
end

% Detect by hit-miss in crisp image
detectByHitMiss = 0;
if detectByHitMiss
    for nLetter = 1:length(letters)
        SE1 = imerode(templates{nLetter}, ones(3,3));
        SE2 = imdilate(templates{nLetter}, ones(5,5)) - ...
            imdilate(templates{nLetter}, ones(3,3));
        imgTestHitMiss = bwhitmiss(imgTestBin, SE1, SE2);
        rows = 50:height-50; cols = 50:width-50;
        if sum(sum(imgTestHitMiss(rows,cols))) > 0
            fprintf('Detected clean, hit-miss: %s\n', letters(nLetter));
            imgTestHitMissDilate = imdilate(imgTestHitMiss, SE1);
            figure(3); clf; set(gcf, 'Color', 'w');
            imshow(max(imgTestHitMissDilate, alpha*imgTestBin));
            pause
        end
    end % nLetter
end

% Load and binarize noisy test image
imgTestNoise = im2double(imread('hw3_license_plate_noisy.png'));
imgTestNoiseGray = rgb2gray(imgTestNoise);
imgTestNoiseBin = double(imgTestNoiseGray < threshold);
figure(4); clf; set(gcf, 'Color', 'w');
imshow(imgTestNoiseBin);

% Detect by hit-miss in noisy image
detectByHitMiss = 1;
if detectByHitMiss
    for nLetter = 1:length(letters)
        SE1 = imerode(templates{nLetter}, ones(3,3));
        SE2 = imdilate(templates{nLetter}, ones(5,5)) - ...
            imdilate(templates{nLetter}, ones(3,3));
        imgTestHitMiss = bwhitmiss(imgTestNoiseBin, SE1, SE2);
        rows = 50:height-50; cols = 50:width-50;
        if sum(sum(imgTestHitMiss(rows,cols))) > 0
            fprintf('Detected noisy, hit-miss: %s\n', letters(nLetter));
            imgTestHitMissDilate = imdilate(imgTestHitMiss, SE1);
            figure(3); clf; set(gcf, 'Color', 'w');
            imshow(max(imgTestHitMissDilate, alpha*imgTestBin));
            pause
```

```
        end
    end % nLetter
end

% Detect by rank filter in noisy image
detectByRankFilter = 1;
if detectByRankFilter
    for nLetter = 1:length(letters)
        SE1 = imerode(templates{nLetter}, ones(3,3));
        SE2 = imdilate(templates{nLetter}, ones(5,5)) - ...
            imdilate(templates{nLetter}, ones(3,3));
        imgTestHitMiss1 = ordfilt2(imgTestNoiseBin, 15, SE1);
        imgTestHitMiss2 = ordfilt2(~imgTestNoiseBin, 12, SE2);
        imgTestHitMiss = min(imgTestHitMiss1, imgTestHitMiss2);
        rows = 50:height-50; cols = 50:width-50;
        if sum(sum(imgTestHitMiss(rows,cols))) > 0
            fprintf('Detected noisy, rank: %s\n', letters(nLetter));
            imgTestHitMissDilate = imdilate(imgTestHitMiss, SE1);
            figure(3); clf; set(gcf, 'Color', 'w');
            imshow(max(imgTestHitMissDilate, alpha*imgTestNoiseBin));
            pause
        end
    end % nLetter
end
```

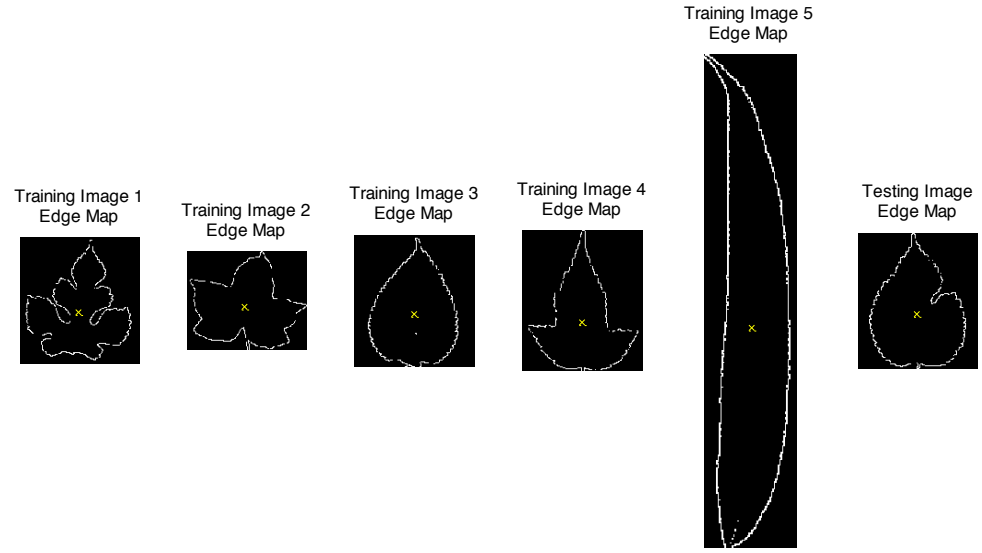
2. Plant Leaf Classification

The algorithm we present here utilizes the shape of each leaf for classification. Other algorithms are also very welcome and given full credit if the correct classification results are achieved.

Step 1:
We binarize each leaf image by thresholding with Otsu's method.

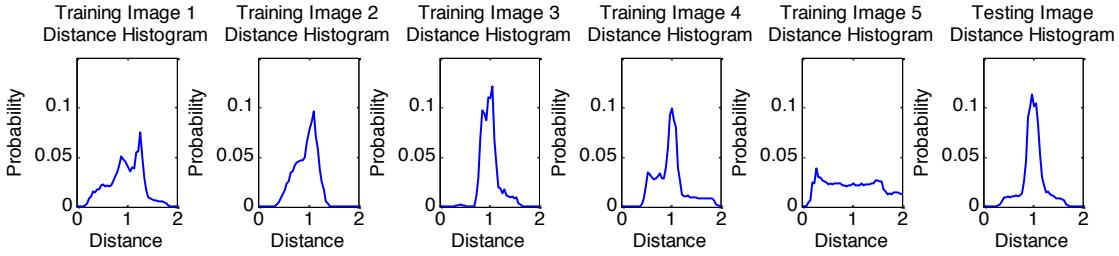


Step 2:
We extract a binary edge map using a morphological edge detector: compute the difference between the binary leaf image and dilate(binary leaf image, 3x3 square SE).



Step 3:
We compute the centroid of each leaf from the binary leaf image. These centroids are shown as yellow X's in the figure above.

Step 4:
We compute the distance between every edge pixel and the leaf centroid. These distances are then normalized by dividing by the median distance. Then, a histogram of the normalized distances is generated. We use 50 uniformly spaced bins in the range [0,2]. To avoid zero-count bins in the histogram, we initialize each bin to have a prior count of 0.5. Each histogram is finally normalized to have a sum of 1, resulting in an empirical probability mass function.



Step 5:
We compute the distance between the testing image's histogram and every training image's histogram. Two different distance measures for histograms are tested: (1) L1 distance and (2) symmetric KL divergence.

$$d_{L1}(p_1, p_2) = \sum_n |p_1(n) - p_2(n)|$$

$$d_{KL}(p_1, p_2) = \sum_n p_1(n) \log\left(\frac{p_1(n)}{p_2(n)}\right) + \sum_n p_2(n) \log\left(\frac{p_2(n)}{p_1(n)}\right)$$
$$= \sum_n \left(p_1(n) - p_2(n)\right)\left(\log p_1(n) - \log p_2(n)\right)$$

Below, we show the distances between the testing image's histogram and every training image's histogram, where it can be seen that the correct training image (#3) has the lowest distance for both L1 and KL.

Training Image	L1 Distance	KL Divergence
1	0.7162	0.6864
2	0.5155	0.4637
3	0.2885	0.2978
4	0.4060	0.3657
5	1.0558	1.7576

MATLAB Code:

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Plant Leaf Classification
% Script by David Chen, Huizhong Chen

clc; clear all;

% Compute signatures
numTrain = 5;
for nImage = 1:numTrain+1
```

```
    % Load image
    if nImage <= numTrain
        img = im2double(imread(sprintf('hw3_leaf_training_%d.jpg', nImage)));
    else
        img = im2double(imread('hw3_leaf_testing_1.jpg'));
    end
    imgGray = rgb2gray(img);

    % Binarize image
    thresh = graythresh(imgGray);
    imgBin = ~im2bw(imgGray, thresh);

    % Extract edge map
    imgBinDil = imdilate(imgBin, ones(3,3));
    imgEdge = double(imgBin ~= imgBinDil);

    % Extract signature
    [signature, bins, centr] = extract_signature(imgBin, imgEdge);
    signatures{nImage} = signature;

    % Show binary image
    figure(1);
    subplot(1,numTrain+1,nImage);
    imshow(imgBin); hold on;
    h = plot(centr(1), centr(2), 'yx');
    set(h, 'MarkerSize', 8);
    set(gca, 'FontSize', 14);
    if nImage <= numTrain
        title({sprintf('Training Image %d', nImage), 'Thresholded Image'});
    else
        title({'Testing Image', 'Thresholded Image'});
    end

    % Show edge map
    figure(2);
    subplot(1,numTrain+1,nImage);
    imshow(imgEdge); hold on;
    h = plot(centr(1), centr(2), 'yx');
    set(h, 'MarkerSize', 8);
    set(gca, 'FontSize', 14);
    if nImage <= numTrain
        title({sprintf('Training Image %d', nImage), 'Edge Map'});
    else
        title({'Testing Image', 'Edge Map'});
    end

    % Show signature
    figure(3);
    subplot(1,numTrain+1,nImage);
    h = plot(bins, signature, 'LineWidth', 2);
    set(gca, 'FontSize', 14);
    axis([min(bins) max(bins) 0 0.15]);
    xlabel('Distance'); ylabel('Probability');
    if nImage <= numTrain
        title({sprintf('Training Image %d', nImage), 'Distance Histogram'});
    else
        title({'Testing Image', 'Distance Histogram'});
    end
end % nImage

% Compute distances between signatures
for nTrain = 1:numTrain
    v1 = signatures{nTrain};
    v2 = signatures{numTrain+1};
    distL1 = sum(abs(v1 - v2));
    distKL = sum( (v1 - v2) .* (log(v1) - log(v2)) );
    fprintf('Training %d: L1 Dist = %.4f\n', nTrain, distL1);
    fprintf('Training %d: KL Dist = %.4f\n', nTrain, distKL);
end % nTrain
```

```
function [signature, bins, centr] = extract_signature(imgBin, imgEdge)

% Find centroid
imgLabel = bwlabel(imgBin);
shapeProps = regionprops(imgLabel, 'Centroid');
centr = shapeProps(1).Centroid;

% Find distances between centroid and edge pixels
[y,x] = find(imgEdge == 1);
dists = zeros(1, numel(y));
for n = 1:numel(y)
    dists(n) = norm(centr - [x(n) y(n)]);
end % n
dists = dists / median(dists);

% Create distance histogram / pmf
bins = linspace(0,2,50);
beta = 0.5;
signature = hist(dists, bins) + beta;
signature = imfilter(signature, [0.125 0.125 0.5 0.125 0.125]);
signature = signature / sum(signature);
```

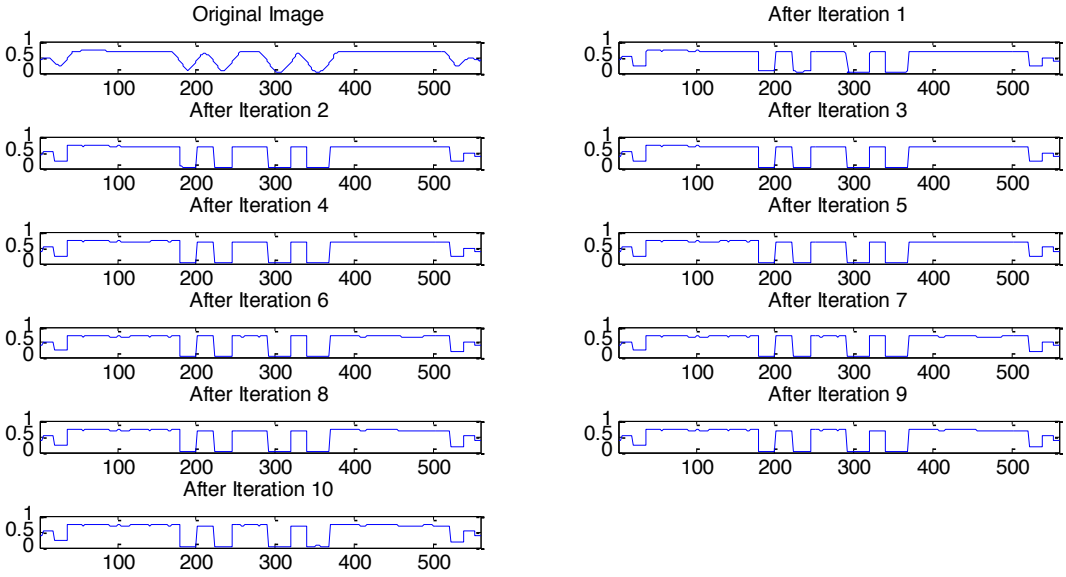
3. Sharpness Enhancement by Dilation and Erosion

Part A:
A circular disk with radius 10 is used as the SE. After running 10 iterations of the algorithm, we obtain the following result. The edges for the people symbol, the text, and the sign's outer boundaries all appear much sharper than in the original image.

Original Image

Processed Image

Part B:
The intensity profile for row 338 is plotted below after different number of iterations. As the algorithm proceeds, the transitions in the profile become increasingly sharper.

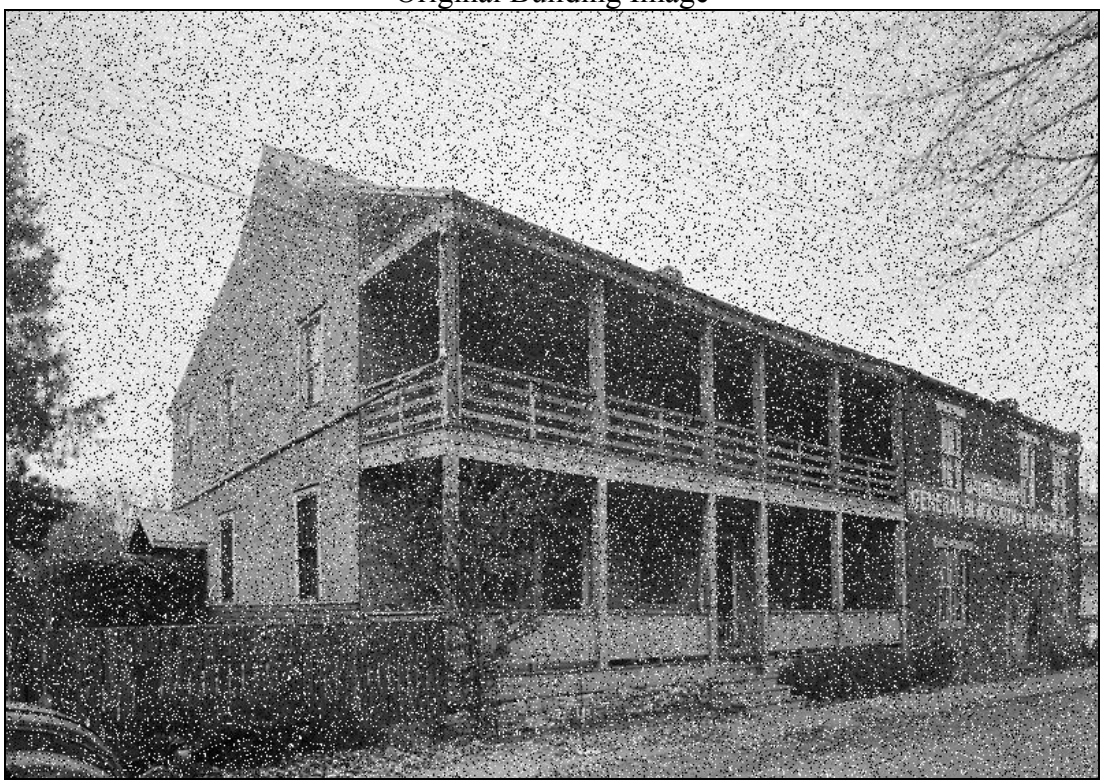


MATLAB Code:

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Sharpness Enhancement
% Script by David Chen, Huizhong Chen

clc; clear all;

% Load image
img = im2double(imread('road_sign_school_blurry.jpg'));
figure(1); clf;
imshow(img);

% Perform iterative enhancement
numIterations = 10;
SE = strel('disk', 10);
row = 338;
rowSamples = img(row,:);
for nIter = 1:numIterations
    imgDilate = imdilate(img, SE);
    imgErode = imerode(img, SE);
    imgMid = 0.5*(imgDilate + imgErode);
    idxDilate = find(img >= imgMid);
    idxErode = find(img < imgMid);
    img(idxDilate) = imgDilate(idxDilate);
    img(idxErode) = imgErode(idxErode);
    rowSamples = [rowSamples; img(row,:)];
end % nIter
figure(2); clf;
imshow(img);
imwrite(img, 'road_sign_school_enhanced.jpg');

% Draw intensity profiles
figure(3); clf;
subplotNum = 1;
for nIter = 1:11
    subplot(6,2,subplotNum);
    plot(rowSamples(nIter,:));
    axis([1 size(img,2) 0 1]);
    if nIter == 1
        title('Original Image');
    else
        title(sprintf('After Iteration %d', nIter-1));
    end
    subplotNum = subplotNum + 1;
end % nIter
figure(4); clf;
plot(rowSamples(1,:)); xlabel('x');
axis([1 size(img,2) 0 1]);
```

4. Noise Reduction by Median Filtering

Original Building Image



Median Filtering, 3x3 Window

Median Filtering, 5x5 Window

Weighted Median Filtering, 5x5 Window

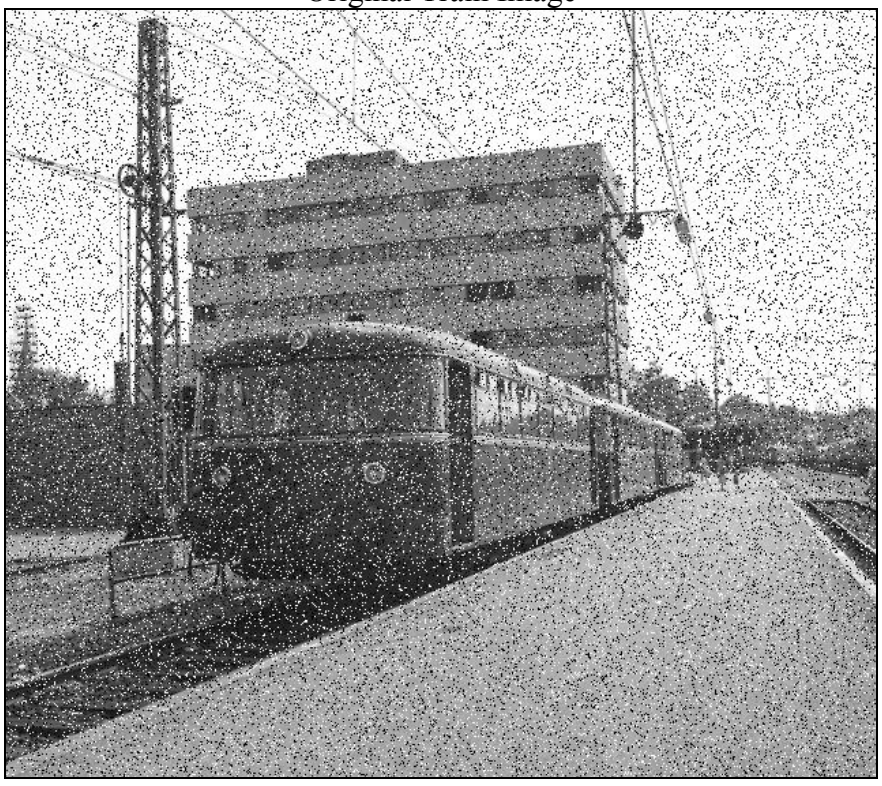
Median Filtering, 3x3

Median Filtering, 5x5

Weighted Median Filtering

For the building image, median filtering with a 3x3 window removes many of the noise specks, but some fairly noticeable noise specks still remain. Median filtering with a 5x5 window reduces the number of noise specks further, but sharp edges and features in the image have been blurred. Finally, weighted median filtering with a 5x5 window achieves the best result: noise specks are almost entirely removed, while sharp edges and features are better preserved.

Original Train Image

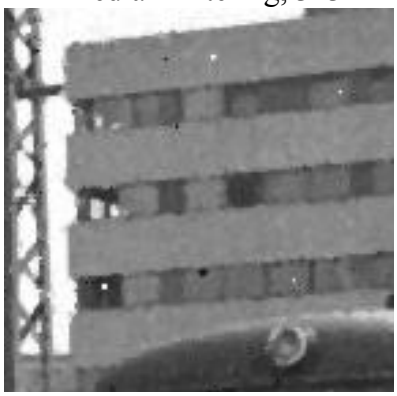


Median Filtering, 3x3 Window

Median Filtering, 5x5 Window

Weighted Median Filtering, 5x5 Window

Median Filtering, 3x3



Median Filtering, 5x5

Weighted Median Filtering

Similarly, for the train image, weighted median filtering with a 5x5 window obtains the best result amongst the three median filtering methods. Noise specks are almost entirely removed while the sharp edges and features in the image are better preserved.

MATLAB Code:

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Median Filtering
% Script by David Chen, Huizhong Chen

clc; clear all;
imageFiles = {'building.jpg', 'train.jpg'};
for nImage = 1:length(imageFiles)
```

```
    % Load image
    img = im2double(imread(imageFiles{nImage}));
    figure(1); clf; imshow(img);
    [path, name, ext] = fileparts(imageFiles{nImage});

    % Perform median filtering
    imgMedFilt = medfilt2(img, [3 3], 'symmetric');
    figure(2); clf; imshow(imgMedFilt);
    imwrite(imgMedFilt, [name '_med_3x3.jpg']);
    imgMedFilt = medfilt2(img, [5 5], 'symmetric');
    figure(3); clf; imshow(imgMedFilt);
    imwrite(imgMedFilt, [name '_med_5x5.jpg']);

    % Perform weighted median filtering
    weights = [ 0 1 1 1 0
                1 2 2 2 1
                1 2 4 2 1
                1 2 2 2 1
                0 1 1 1 0 ];
    imgWeMedFilt = weighted_median_filter(img, weights);
    figure(4); clf; imshow(imgWeMedFilt);
    imwrite(imgWeMedFilt, [name '_we_med_5x5.jpg']);
end % nImage

function imgOut = weighted_median_filter(imgIn, weights)

% Check parameters
[h, w, c] = size(imgIn); [hw, ww] = size(weights);
hhw = floor(hw/2); hww = floor(ww/2);
if c ~= 1
    error('weighted_median_filter only works for grayscale images');
end
if (mod(hw,2) == 0) || (mod(ww,2) == 0)
    error('weights array must have odd dimensions');
end

% Perform weighted median filtering
imgOut = imgIn;
for y = 1:h
    if mod(y,100) == 0
        disp(sprintf('Processing row %d/%d', y, h));
        pause(0.1);
    end
    for x = 1:w
        % Accumulate samples
        samples = [];
        for xp = max(1,x-hhw):min(w,x+hhw)
            dx = xp - x;
            dxw = dx + hww + 1;
            for yp = max(1,y-hww):min(h,y+hww)
                dy = yp - y;
                dyw = dy + hhw + 1;
                if weights(dyw, dxw) > 0
                    samples = [samples imgIn(yp,xp)*ones(1,weights(dyw,dxw))];
                end
            end % yp
        end % xp

        % Calculate median
        imgOut(y,x) = median(samples);
    end % x
end % y
```

Bonus #1: Android with OpenCV

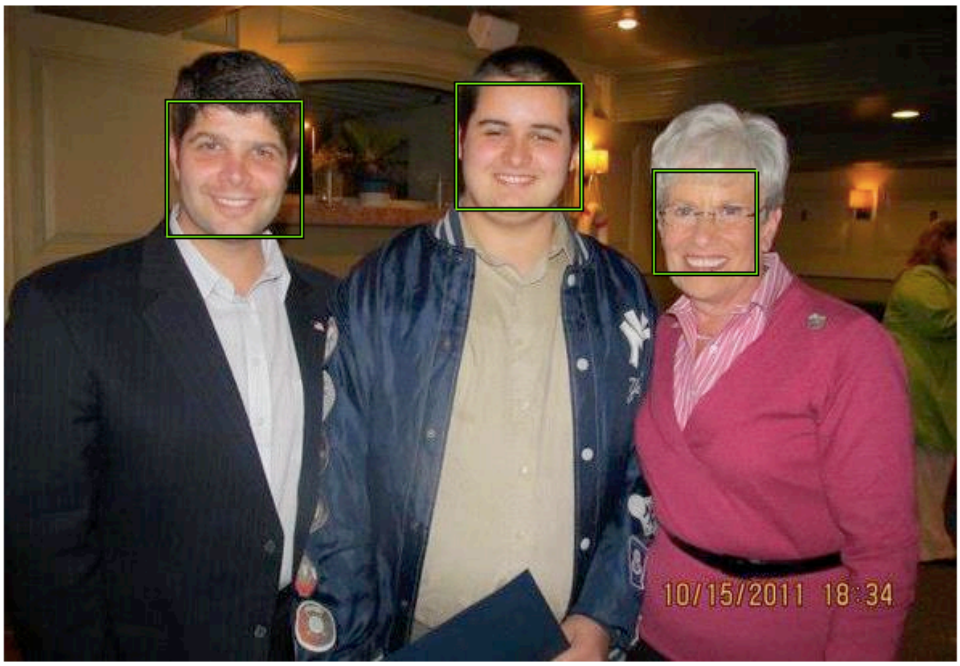
Tutorial #2 (OpenCV for Android Setup) explains how to install the OpenCV programming tools and build applications that call OpenCV functions.

Here is a screenshot of the application running on the Droid showing MSERs overlaid. You might have chosen to test a different application for this problem—edges/lines/circles detection, locally adaptive binarization, human face detection—and would receive full credit for showing a screenshot that clearly demonstrates the underlying algorithm being applied.

Bonus #2: OpenCV Library

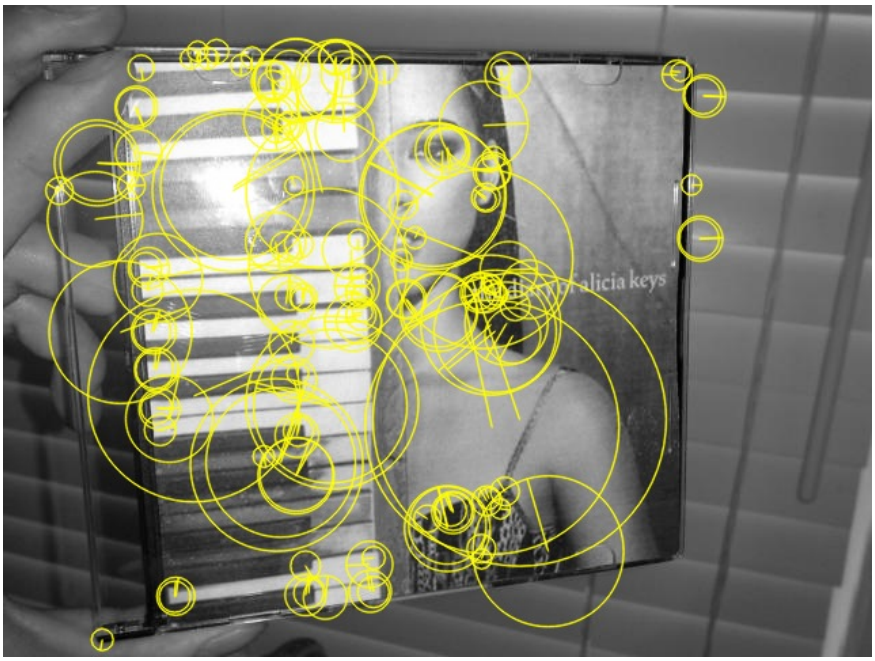
Face Detection

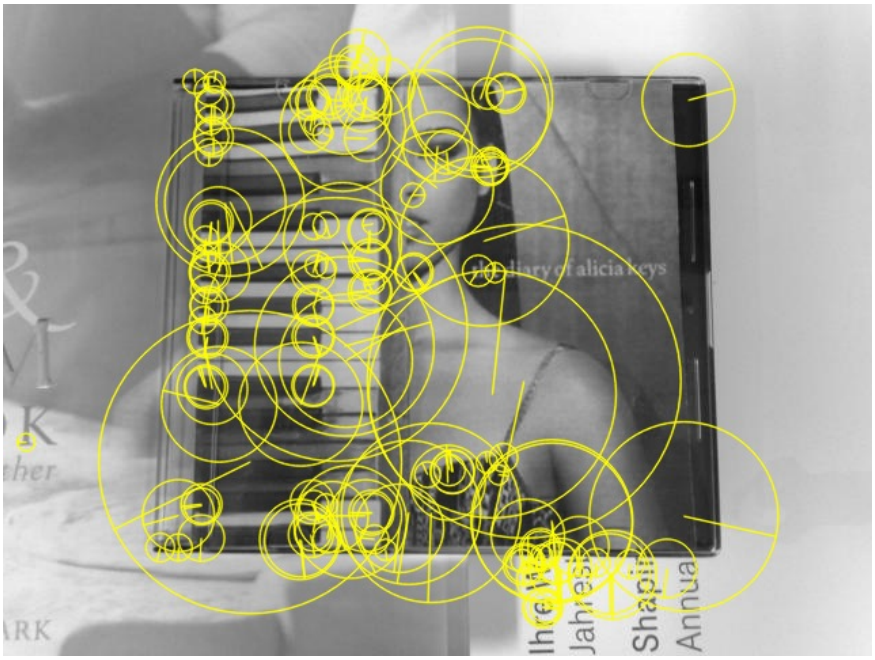
The images below show the faces detected by the Viola-Jones face detector. We can directly call the face detector within MATLAB using the MEX interface provided by the mexopencv toolkit.



SURF Keypoint Extraction

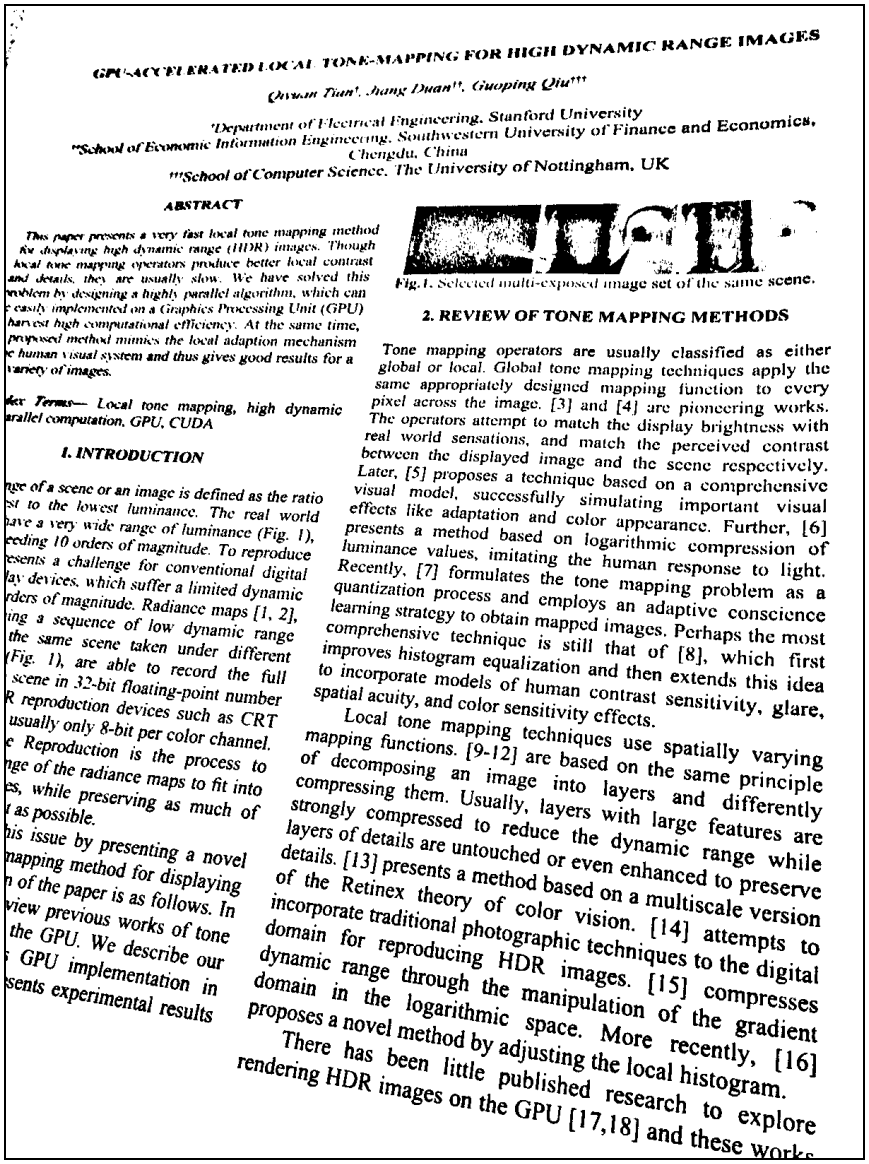
The images below show the detected SURF keypoints. These keypoints are detected at various scales and each has a local orientation. By inspection, many keypoints are detected in common on the CD cover in the two images.





Adaptive Thresholding

The image below shows the result of adaptive thresholding. We used a block size of 61x61. Despite the severely non-uniform illumination in the original grayscale image, nearly all of the text is correctly set to black pixels and all of the blank space on the page is correctly set to white pixels.



GPU-ACCELERATED LOCAL TONE-MAPPING FOR HIGH DYNAMIC RANGE IMAGES

Qiwan Tian[1], Jiang Duan[11], Guoping Qiu[111]

[1]Department of Electrical Engineering, Stanford University
[11]School of Economic Information Engineering, Southwestern University of Finance and Economics, Chengdu, China
[111]School of Computer Science, The University of Nottingham, UK

ABSTRACT

This paper presents a very fast local tone mapping method for displaying high dynamic range (HDR) images. Though local tone mapping operators produce better local contrast and details, they are usually slow. We have solved this problem by designing a highly parallel algorithm, which can be easily implemented on a Graphics Processing Unit (GPU) to harvest high computational efficiency. At the same time, the proposed method mimics the local adaption mechanism of the human visual system and thus gives good results for a variety of images.

Index Terms— Local tone mapping, high dynamic range, parallel computation, GPU, CUDA

1. INTRODUCTION

The range of a scene or an image is defined as the ratio of the highest to the lowest luminance. The real world scenes have a very wide range of luminance (Fig. 1), often exceeding 10 orders of magnitude. To reproduce this presents a challenge for conventional digital display devices, which suffer a limited dynamic range of orders of magnitude. Radiance maps [1, 2], created using a sequence of low dynamic range images of the same scene taken under different exposures (Fig. 1), are able to record the full dynamic range of a scene in 32-bit floating-point number precision. HDR reproduction devices such as CRT monitors usually only 8-bit per color channel. Tone Reproduction is the process to compress the range of the radiance maps to fit into the devices, while preserving as much of the contrast as possible.

This paper addresses this issue by presenting a novel high dynamic range tone mapping method for displaying images. The rest of the paper is as follows. In Section 2 we review previous works of tone mapping operators. Section 3 describes our implementation on the GPU. We describe our GPU implementation in Section 4. Section 5 presents experimental results and Section 6 concludes the paper.

Fig.1. Selected multi-exposed image set of the same scene.

2. REVIEW OF TONE MAPPING METHODS

Tone mapping operators are usually classified as either global or local. Global tone mapping techniques apply the same appropriately designed mapping function to every pixel across the image. [3] and [4] are pioneering works. The operators attempt to match the display brightness with real world sensations, and match the perceived contrast between the displayed image and the scene respectively. Later, [5] proposes a technique based on a comprehensive visual model, successfully simulating important visual effects like adaptation and color appearance. Further, [6] presents a method based on logarithmic compression of luminance values, imitating the human response to light. Recently, [7] formulates the tone mapping problem as a quantization process and employs an adaptive conscience learning strategy to obtain mapped images. Perhaps the most comprehensive technique is still that of [8], which first improves histogram equalization and then extends this idea to incorporate models of human contrast sensitivity, glare, spatial acuity, and color sensitivity effects.

Local tone mapping techniques use spatially varying mapping functions. [9-12] are based on the same principle of decomposing an image into layers and differently compressing them. Usually, layers with large features are strongly compressed to reduce the dynamic range while layers of details are untouched or even enhanced to preserve details. [13] presents a method based on a multiscale version of the Retinex theory of color vision. [14] attempts to incorporate traditional photographic techniques to the digital domain for reproducing HDR images. [15] compresses dynamic range through the manipulation of the gradient domain in the logarithmic space. More recently, [16] proposes a novel method by adjusting the local histogram.

There has been little published research to explore rendering HDR images on the GPU [17,18] and these works

MATLAB Code:

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Face Detection
% Script by David Chen, Huizhong Chen

clc; clear all;

mexopencvDir = '../../../mexopencv-master';
addpath(mexopencvDir);
```

```
% Load a face detector
detector = cv.CascadeClassifier([mexopencvDir '/test/haarcascade_frontalface_alt2.xml']);

% Process images
imageFiles = {'faces_1.jpg', 'faces_2.jpg'};
for nImage = 1:length(imageFiles)
    % Load image
    im = imread(imageFiles{nImage});

    % Preprocess
    gr = cv.cvtColor(im, 'RGB2GRAY');
    gr = cv.equalizeHist(gr);

    % Detect
    boxes = detector.detect(gr, 'ScaleFactor',  1.3, ...
                                'MinNeighbors', 4, ...
                                'MinSize',      [30, 30]);
    % Draw results
    figure(1); clf;
    imshow(im);
    for i = 1:numel(boxes)
        rectangle('Position',  boxes{i} - [1 1 0 0], ...
                  'EdgeColor', 'k');
        rectangle('Position',  boxes{i} + [1 1 0 0], ...
                  'EdgeColor', 'k');
        rectangle('Position',  boxes{i}, ...
                  'EdgeColor', 'g');
    end % i

    if nImage < length(imageFiles)
        pause;
    end
end % nImage
```

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Feature Matching
% Script by David Chen, Huizhong Chen

clc; clear all;

mexopencvDir = '../../../mexopencv-master';
addpath(mexopencvDir);

% Load images
im1 = imread('hw3_cd_cover_1.jpg');
im1 = cv.cvtColor(im1,'RGB2GRAY');
im2 = imread('hw3_cd_cover_2.jpg');
im2 = cv.cvtColor(im2,'RGB2GRAY');

% Detect SURF keypoints
detector = cv.FeatureDetector('SURF');
keypoints1 = detector.detect(im1);
keypoints2 = detector.detect(im2);

% Retain keypoints with strongest responses
keypoints1 = keypoints1(1:150);
keypoints2 = keypoints2(1:150);

% Visualize keypoints
figure(1); clf;
subplot(1,2,1);
im1_keypoints = cv.drawKeypoints(im1, keypoints1, ...
    'DrawRichKeypoints', 1, 'Color', [255 255 0 0]);
imshow(im1_keypoints);
subplot(1,2,2);
im2_keypoints = cv.drawKeypoints(im2, keypoints2, ...
    'DrawRichKeypoints', 1, 'Color', [255 255 0 0]);
```

```
imshow(im2_keypoints);
```

```
% EE368/CS232, Spring 2013-2014
% Homework 3
% Problem: Adaptive Thresholding
% Script by David Chen, Huizhong Chen

clc; clear all;

mexopencvDir = '../../../mexopencv-master';
addpath(mexopencvDir);

% Load image
im = imread('hw3_paper.png');

% Perform adaptive thresholding
imBin = cv.adaptiveThreshold(im, 255, ...
    'AdaptiveMethod', 'Mean', 'ThresholdType', 'Binary', ...
    'BlockSize', 61, 'C', 20);

% Show binarized result
figure(1); clf;
imshow(imBin);
imwrite(imBin, 'result.png');
```