Real-time texture boundary detection from ridges in the standard deviation space

Ray Hidayat
ray.hidayat@pg.canterbury.ac.nz
Richard Green
richard.green@canterbury.ac.nz

Computer Science and Software Engineering Department
University of Canterbury Private Bag 4800, Christchurch 8140, New Zealand

Abstract

This paper presents a novel texture boundary detector called the standard deviation ridge detector. At 43.29 frames per second, it is one of the few texture boundary detectors that can run in realtime. With its Berkeley segmentation benchmark F-statistic of 0.62, the algorithm outperforms all existing realtime texture boundary detectors. The use of the boundary detector would induce noticable improvements to all realtime machine vision applications. In addition, a ridge detector, which is better suited to realtime than other approaches, is presented as part of the proposed algorithm.

1 Introduction

The past decade has seen great advancements in the field of texture analysis. Categorically, many of these advancements have been driven by the invention of the texton approach to texture analysis. At its core, this approach uses high-dimensional analysis of texture, where the dimensions correspond to convolved filter responses. The development of textons has lead to state-of-the-art developments in areas such as segmentation and classification. Refer to [14] for a detailed description of this texton approach.

Unfortunately, the multiple convolutions (often 25 or more) and high-dimensional analysis required by the texton approach make it too slow to run in realtime. Consequently, most of the gains from the past decade of texture research have been inapplicable to realtime applications. For this reason, this paper proposes a fast texture boundary detection algorithm that, most importantly, is able to run in realtime.

This paper proposes the standard deviation ridge detector, a realtime texture boundary detector that, as its name suggests, works by detecting ridges in an image's standard deviation space. There are two primary contributions made by this paper. Firstly, this paper presents a new ridge detection algorithm that is more useful than existing ridge detection methods. Secondly, the proposed texture boundary detector itself outperforms all existing realtime texture boundary detectors.

BMVC 2009 doi:10.5244/C.23.5

2 Related work

2.1 Realtime texture boundary detectors

Many texture boundary detectors exist, however almost all of them do not run in real-time. The following three algorithms represent the best and most significant.

The simplest approach is **anisotropic surround suppression** [3]. This algorithm works on the simple predicate that if an edge is surrounded by other edges of similar strength, then it belongs to a repeating texture and should therefore be suppressed. This can be implemented very fast as it consists of only two operations – a smooth and a subtraction. This extremely fast algorithm only produces mediocre results however, which has lead to attempts being made to improve its quality [11]. The algorithm proposed by this paper significantly improves on the anisotropic surround suppression method in terms of quality, but not execution speed.

The second approach is the **second moment matrix** [6, 9, 10]. This method detects texture boundaries by identifying areas of locally unbalanced/anisotropic gradients, which works because gradients tend to be isotropic (the same in all directions) within a repeating texture. The second moment matrix is one of the best approaches for realtime texture boundary detection, and is popular because it serves as the basis for widely-known methods like the Harris-Stephens corner detector [4]. Similarly, the proposed algorithm identifies locally unbalanced gradients. However, because the proposed algorithm analyses the standard deviation space gradients instead of brightness gradients, it produces better results.

The convolved variance ridges algorithm [5] appears to be the best in realtime texture boundary detection to date. Both this algorithm and the proposed algorithm are variance-based. But unlike the convolved variance ridges algorithm, the proposed algorithm uses unweighted rectangular smoothing and other alternatives instead of convolution. This not only means the proposed algorithm runs over twice as fast, but because the unweighted averages are actually better at representing texture, the quality of its results are also better.

2.2 Edge detectors

It is important to make the distinction between an edge detector and a boundary detector. An edge detector, such as the widely used Canny edge detector [1], identifies low level changes in the image. Often, this is not useful when dealing with real-world images. Martin, Fowlkes and Malik [9] stated the problem well when they said: "The Canny edge detector fires wildly inside textured regions where high-contrast edges are present, but no boundary exists." Boundary detectors solve this problem because they work at a higher-level than edge detectors – they attempt to find only the important divisions in the image, and unlike edge detectors, they suppress all the unimportant ones. One of the reasons the proposed algorithm is so useful is because it is a boundary detector and not an edge detector, as demonstrated by the results in section 4.

2.3 Standard deviation texture analysis

The algorithm proposed by this paper uses standard deviation to identify texture boundaries. As standard deviation is such a widely-known statistic, it has already been used in the past for texture-aware algorithms such as segmentation by weighted aggregation [13] and an edge-preserving smoothing filter [12]. The proposed work uses standard deviation similarly.

2.4 Ridge detection algorithms

At its heart, the proposed algorithm is a ridge detection algorithm. Principally, existing ridge detection algorithms can be categorised by whether they are designed to work on binary images, or greyscale images. Both approaches have shortcomings and so are not useful for realtime texture boundary detection.

Binary ridge detection involves applying repeated morphological operations, one good example being the Canny edge detector's hysteresis stage [10]. This requires a binary image, which is where the problem lies. This approach would be likely to introduce a sensitive thresholding parameter, as well as unintended thresholding artefacts. The quality of the proposed algorithm would be compromised by such an approach.

In general, greyscale ridge detection algorithms require the gradients of an image to be calculated. The gradients are used to locate one-dimensional local maxima in an image, which are then accentuated, the result being that the ridges in the image are detected [7, 9]. The problem with this is, calculating the first and second derivatives of an image at multiple orientations would involve convolving the image many times with the first-derivative Gaussian kernel. As stated previously, convolutions are slow, and faster approximations can be used when time is critical. The proposed algorithm does not use convolution for its ridge detection, making it many times faster than existing algorithms.

With existing ridge detection algorithms being inadequate for the purpose of realtime texture boundary detection, a new algorithm has been developed, which is one of the main contributions of this paper.

3 Algorithm

Fundamentally, the proposed algorithm works by detecting ridges in the standard deviation space of an image. Standard deviation is chosen for the following three reasons:

1. Unlike most features of texture, standard deviation can be calculated fast.

2. Standard deviation can detect texture boundaries (where two textures meet). This follows logically. If only one texture was in the local neighbourhood, the standard deviation only has to encapsulate the intra-class variation. When a second texture is present, the standard deviation must represent the inter-class variation in addition to the intra-class variation. This means that, standard deviation will peak wherever two or more textures meet, as long as there is sufficient inter-class variation.

3. Every texture boundary detector also needs a way to suppress the intra-class variations within a texture from appearing as boundaries. Standard deviation can achieve this as well, because in general, for different areas within the same texture, standard deviation tends to be approximately equal. This allows for a properly designed and tuned algorithm to detect different areas of the same texture.

These triple benefits make standard deviation an excellent choice for a real-time algorithm, and an illustration of this can be seen in figure 1. It must be noted that not all forms of texture can be distinguished solely by standard deviation, but this is acceptable when the primary goal is a realtime algorithm.

An important consideration for the proposed algorithm is that it is intended to be used as a preprocessor to another realtime algorithm. This imposes a number of constraints on

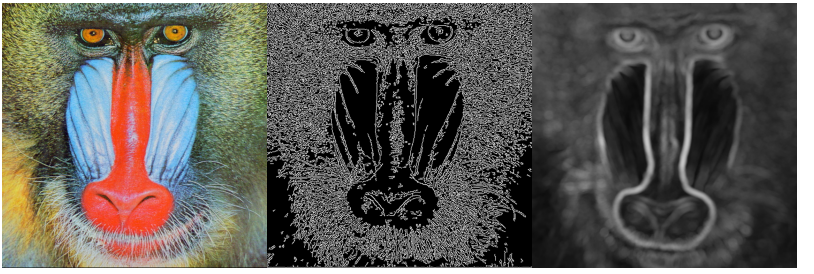


Figure 1: The standard deviation transform (right) of an image (left). The two most important benefits of choosing standard deviation are clear: the boundaries are detected, while the repeated variations in the texture are produce little response. The Canny edge map (middle) is shown to emphasise how good standard deviation is at suppressing texture edges.

its design. First, it must run faster than realtime, so as to leave as much time as possible for the higher-level algorithm. Second, because the classification stage (which of course is very important) will be part of the higher-level algorithm, it is not included as part of this algorithm. It must be strongly emphasized that this is why a machine learning/classification stage is not presented as part of the algorithm in this paper. The proposed algorithm also already outperforms all existing realtime methods, and so by induction, the argument is that the proposed algorithm will continue to outperform them should a machine learning stage be added to postprocess the algorithm's results.

The proposed algorithm can be expressed by the equation 1:

$$b_k(I) = r_k(g_k^s(I)) - \|g_k^s(I)\| \quad \text{where} \quad g_k^s(I) = g_k(s_k(I)) \tag{1}$$

The output of each of these stages is shown in figure 3. In equation 1, the algorithm begins with an $m \times n$ image I, which would be presented to the algorithm in the CIELAB colour space. In this paper, section 3.1 describes $s_k(.)$, which calculates the standard deviation transform of the image. Section 3.2 describes $g_k(.)$, which calculates the gradients when given the standard deviation image. For convenience, the name $g_k^s(.)$ is given for succession of these two operations. $r_k(.)$ identifies the ridges from the gradient image. Finally, the final boundaries are detected by subtracting the gradient magnitude image $\|g_k^s(I)\|$ from the ridge image. Both of these last two operations will be described in section 3.3. The proposed algorithm only takes one parameter, the window size parameter k, which in essence determines the cutoff wavelength between when variations should be considered texture (and would therefore be suppressed) as opposed to when they should be considered a boundary (and would therefore be emphasised). The parameter is intuitive, as illustrated in figure 2.

3.1 Standard deviation transform

The algorithm begins by calculating the standard deviation of each pixel neighbourhood, in accordance with equation 2:

$$s_k(I) = \sqrt{\|\mu_k(I^2) - \mu_k(I)^2\|} \tag{2}$$

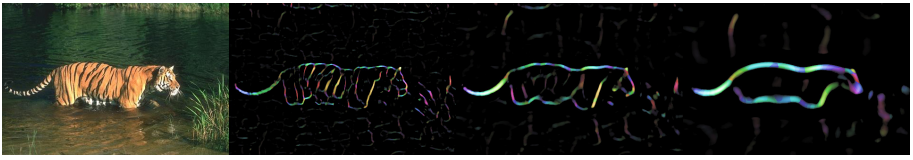


Figure 2: The effect of k on the proposed standard deviation ridge algorithm, arranged as follows: original image (far left), $k = 4$ (middle left), $k = 8$ (middle right) and $k = 16$ (far right). Hue represents boundary orientation.

$\mu_k(.)$ is the smoothing operator, calculated as the local mean within a $(2k+1) \times (2k+1)$ rectangular neighbourhood centered on each pixel. As each channel would be smoothed separately, the L2 norm is used to combine the three channels into a single standard deviation measure.

3.2 Gradient vector transform

Next, the gradient image is calculated from the standard deviation image. The purpose for this stage is as follows. Standard deviation is approximately equal at different areas in the same texture. This means that textured areas will have very little gradient in the standard deviation space, and instead, gradients will generally only be present on texture boundaries. So by executing this stage, the repeating variations within texture can be suppressed, and texture boundaries can be enhanced. Secondarily, calculating the gradient is a necessary step in order to perform ridge detection.

This stage involves calculating the gradient vector individually for each pixel neighbourhood. The term **gradient vector** is used because both the gradient strength and direction are calculated. This can be formulated as follows:

$$g_k(S) = \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,n} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ g_{m,1} & g_{m,2} & \cdots & g_{m,n} \end{bmatrix} \tag{3}$$

$$g_{x,y} = \sum_{(p,q)\in\Omega} \left(\bar{S}_{(x+p),(y+q)} - \bar{S}_{(x-p),(y-q)}\right) \times \begin{pmatrix} p \\ q \end{pmatrix} \tag{4}$$

where

$$\begin{aligned} &\bar{S} = \mu_k(S) \\ &\Omega = \left\{ \begin{pmatrix} p_1 \\ q_1 \end{pmatrix}, \begin{pmatrix} p_2 \\ q_2 \end{pmatrix}, \ldots, \begin{pmatrix} p_\omega \\ q_\omega \end{pmatrix} \right\} \\ &p_i = \text{round}\left(k\cos\left(\frac{i\pi}{\omega}\right)\right) \quad \text{and} \quad q_i = \text{round}\left(k\sin\left(\frac{i\pi}{\omega}\right)\right) \end{aligned} \tag{5}$$

Equation 3 expresses that each individual pixel is considered individually. Equation 4 calculates the gradient vector for a single pixel. In this equation, the symbol S represents the standard deviation image, which would have been calculated during the previous stage of the

Figure 3: The proposed standard deviation ridge detector at different stages. For the three right panels, hue represents gradient/ridge/boundary orientation. $k = 8$ in all cases.

algorithm. Basically, it calculates the one-dimensional gradients at a number of orientations (determined by ω), and takes the vector sum. The resulting gradient vector will be directed along the principal positive direction of the gradient.

Tests have shown that it is effective to always use $\omega = 8$ in all cases, as this consistently produces good results. All calculations in this paper use $\omega = 8$ for this reason. It is not unusual for a parameter like this to be fixed, the popular Canny edge detector [1] similarly limits its calculations to four orientations.

The smoothed version $\bar{S}$ of the standard deviation image S is used in equation 4. This requires some explanation. It can be seen in equation 4 that each one-dimensional gradient is only calculated from two points on $\bar{S}$. This calculation runs very fast because only two points are used, and allows the cost of calculating $\bar{S}$ to be amortised over several usages during the evaluation of equation 4. The reason $\bar{S}$ is used is, because each pixel in $\bar{S}$ represents the mean of its respective pixel neighbourhood, sampling only two points in $\bar{S}$ will produce a robust estimation of the gradient in that direction. If $\bar{S}$ were not used, sampling such a few number of points would be likely to introduce noise and unreliable results into the algorithm.

3.3 Ridge detection

The previous stage detects gradients only. This is inadequate because, as observed by many researchers before [9], each ridge will produce a double response in the gradient space – one response for the negative gradient on one side of the ridge, and another response for the positive gradient on the other side. This can be seen in the gradient image in figure 3. This stage combines the double gradient responses, so that each ridge produces only a single response.

$$r_k(G) = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{m,1} & r_{m,2} & \cdots & r_{m,n} \end{bmatrix} \tag{6}$$

$$r_{x,y} = \max_{(p,q)\in\Omega} \sqrt{\left[-G_{(x+p),(y+q)} \bullet G_{(x-p),(y-q)}\right]^+} \tag{7}$$

G represents the gradient vector image, which would have been calculated by the previous stage. Ω was defined in equation 5. The $[.]^+$ operator denotes a positive bounding function, which constrains values to the range zero or above. That is, any value less than zero is replaced with zero. Finally, the $\bullet$ operator denotes the dot product.

Similar to the previous stage, this stage considers each pixel individually, as expressed by equation 6. At each particular pixel, equation 7 measures the ridge strength of a number of orientations, and returns the strongest overall response. The ridge strength for a particular orientation is measured by calculating how strongly the gradients on either side of the pixel point towards each other, as this is the defining characteristic of a ridge. The dot product and square root make this calculation in essence like an oriented geometric mean, which conceptually means that there will be no response if, instead of two opposing gradients on either side of the ridge, there is only one gradient present. This stops false positives from occuring.

The result of $r_k(.)$ is a ridge strength image. It turns out that, normally, ridges are overdetected in $r_k(.)$. That means, they appear thicker than they actually are, and sometimes, particularly around corners, a single ridge can be detected multiple times at slightly different locations. An illustration of this can be seen in figure 3. All of this occurs because the ridge strengths are calculated from small samples of only two points each (see equation 7). The most obvious way to solve this would be to use a larger sample size in the calculation of the ridge strengths. Although this would solve the problem, it would also slow down the algorithm. In the following paragraphs, this paper proposes a faster approach that achieves the same purpose.

Equation 1, which defines the overall algorithm, shows that the final step is the subtraction of the gradient magnitude image $\|g_k^s(I)\|$ from the ridge image $r_k(I)$. This subtraction removes many of the overdetected ridges from the ridge image. The logic for this is as follows. A ridge can only exist *between* gradients – it must have a positive gradient on one side, and a negative gradient on the other. Therefore, a ridge cannot be located at the same position as a gradient. By subtracting the gradient image from the ridge image, any ridges that exist where gradients exist are removed.

This is highly effective due to the intersection of two axioms: (1) every ridge is surrounded by gradients, and (2) any mislocalisation of a ridge will occur in its surrounding area. Therefore, because of (1) and (2), practically all of the unfavourable ridge responses are removed. The result is a clean ridge detection which can be calculated very fast. This final ridge detection is returned as the final boundary detection of the algorithm.

4 Results

The proposed standard deviation ridge algorithm was implemented as a single-threaded C++ program. Tests show that the proposed algorithm has produced many highly favourable results. A few examples of this can be seen in figure 4.

Unless otherwise stated, the window size parameter $k = 8$ in all illustrations in this paper. This value was empirically chosen as it generally produced good results for the size of images in the test set. A likely explanation for this is, setting $k = 8$ sets the window size to be 15 $(= 2k + 1)$, and texture in 320×240 images normally has a wavelength of 15 pixels or less. Therefore it is effectively detected by this window size. It is also important to point out that the choice of k has no effect on the speed of the algorithm.

4.1 Speed results

When tested on 2594 320×240 frames, the proposed algorithm's mean run time was 0.0231 seconds, and so can therefore process 43.29 frames per second. Even with this highly

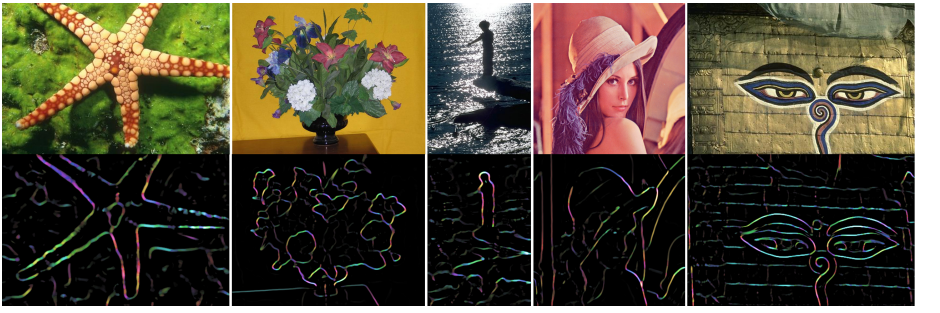
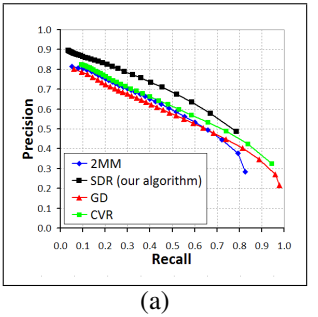
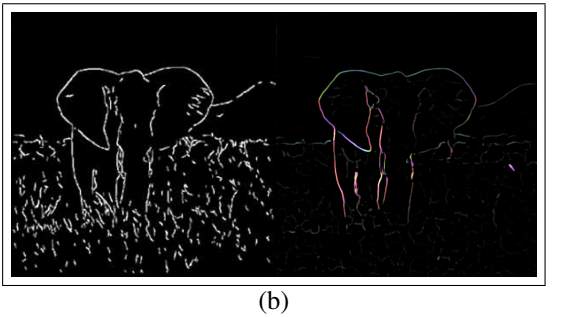


Figure 4: Example results (bottom row) of the proposed standard deviation ridge algorithm on some images (top row). Hue represents boundary orientation and $k = 8$ in all cases.



(a)



(b)

Figure 5: (a) The Berkeley segmentation benchmark precision-recall curves of the proposed standard deviation ridge (SDR) detector with $k = 8$, against the second moment matrix (2MM) with $\sigma = 2$, a Gaussian derivative (GD) with $\sigma = 2$, and the convolved variance ridges (CVR) algorithm with $\sigma = 7/6$. All parameters were optimised for the Berkeley benchmark. (b) SDR algorithm (right) compared to the traditional realtime texture boundary detector – anisotropic suppression (left). For comparison, this is the same image used by [3] and [11]. In the result image for the SDR algorithm, hue represents boundary orientation.

favourable result, there is always the possibility that further optimisations could have been made to the code to improve its speed further. This measurement was taken using an Intel Core 2 Duo 2.66 Ghz machine. Additionally, the time required to capture the image from the camera was not included to ensure that only the speed of the proposed algorithm was being measured.

4.2 Berkeley segmentation benchmark results

To objectively demonstrate the quality of the proposed algorithm, this paper illustrates the algorithm's performance on the Berkeley segmentation benchmark [8]. This benchmark works by algorithmically (and therefore objectively) comparing a segmentation algorithm's result to human-defined ground truths.

The performance of algorithms on the Berkeley segmentation benchmark can be compared by their precision-recall curves, figure 5 shows the relevant ones. Figure 5 clearly

shows the proposed standard deviation ridge algorithm outperforming all other algorithms with its precision-recall.

Another way to compare the precision-recall curves is to use the F-statistic – a single value that sums up an algorithm's entire performance on the Berkeley dataset. The human ground-truths establish that a perfect algorithm would score an F-statistic of 0.79. The worst-case score can be found by using a random number generator for boundary detection, which achieves a score of 0.41. Standard deviation ridge detection scores an F-statistic of 0.62. This can be compared to the second moment matrix, which achieves 0.57, and the convolved variance ridge algorithm, which scores 0.59. These results demonstrate that the proposed algorithm improves on all prior realtime texture boundary detectors.

Although there are algorithms that score higher on the Berkeley benchmark, all of them require much more time to execute. For example, boosted edge learning [2] takes around 12 seconds per image, gaining a F-statistic of 0.66 on the same Berkeley benchmark. Being upward of 500 times slower however, such approaches are not intended for realtime applications, and so cannot be compared to a realtime context. The proposed algorithm substantially outperforms these approaches on speed.

5 Conclusion

From the results it can be seen that the proposed standard deviation ridge detector has a number of substantial improvements over previous work. Unlike most texture boundary detectors, the proposed algorithm runs faster than realtime, with a measured speed of 43.29 frames per second. It also produces better quality results than existing realtime texture boundary detectors with its Berkeley benchmark F-statistic of 0.62. In particular, it even outperforms the state-of-the-art algorithms such as the second moment matrix and the convolved variance ridges algorithm, which can seen clearly from the precision-recall curves in figure 5. Additionally, the proposed algorithm is a boundary detector, not an edge detector, and so it suppresses unimportant texture edges which are unavoidable when using just a simple edge detector. Finally, the proposed algorithm requires only one parameter, which is an intuitive window size parameter. This makes the algorithm easy to use.

Secondarily, the favourable results of the standard deviation ridge detector have proven the worth of the novel ridge detection algorithm that it uses. The ridge detection algorithm (a) does *not* require a binary image, meaning no unnecessary artefacts or parameters are introduced by a thresholding stage, and (b) is faster than existing greyscale ridge detection algorithms because it does not require convolution. Future work could involve applying this ridge detection algorithm to other fields.

The proposed standard deviation ridge algorithm can be used as a preprocessing stage to improve realtime vision applications by making them texture-aware. It is one of the few realtime boundary detectors available, meaning it is able to suppress unimportant texture variations while emphasising important texture boundaries. The algorithm's Berkeley benchmark F-statistic score of 0.62 and its realtime execution speed of 43.29 frames per second fully back up this claim.

References

[1] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[2] P. Dollar, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, 2006.

[3] C. Grigorescu, N. Petkov, and M. A. Westenberg. Contour detection based on non-classical receptive field inhibition. *IEEE Transactions on Image Processing*, 12(7): 729–739, 2003.

[4] C. Harris and M. Stephens. A combined corner and edge detector. *Journal on Data Semantics*, 15:50, 1988.

[5] R. Hidayat and R. Green. Texture-suppressing edge detection in real-time. In *IVCNZ08*, 2008.

[6] S. Konishi, A. L. Yuille, J. Coughlan, and S. C. Zhu. Fundamental bounds on edge detection: An information theoretic evaluation of different edge cues. In *CVPR*, pages 573–579, 1999.

[7] T. Lindeberg. Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, 30(2):117–154, 1998.

[8] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. *ICCV*, 2001.

[9] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 530–549, 2004.

[10] M. Nitzberg and T. Shiota. Nonlinear image filtering with edge and corner enhancement. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(8):826–833, 1992.

[11] G. Papari, P. Campisi, N. Petkov, and A. Neri. A biologically motivated multiresolution approach to contour detection. *Eurasip Journal on Advances in Signal Processing*, 2007.

[12] G. Papari, N. Petkov, and P. Campisi. Artistic edge and corner enhancing smoothing. *IEEE Transactions on Image Processing*, 16(10):2449–2462, 2007.

[13] E. Sharon, A. Brandt, and R. Basri. Segmentation and boundary detection using multiscale intensity measurements. In *CVPR*, 2001.

[14] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context. *International Journal of Computer Vision*, 2006.