Efficient Computation of Adaptive Threshold Surfaces for Image Binarization

Ilya Blayvas, Alfred Bruckstein and Ron Kimmel
Computer Science Department, Technion Institute of Technology
Haifa, ISRAEL, 32000
{blayvas,freddy,ron}@cs.technion.ac.il

Abstract

The problem of binarization of gray level images acquired under nonuniform illumination is reconsidered. Yanowitz and Bruckstein proposed to use for image binarization an adaptive threshold surface, determined by interpolation of the image gray levels at points where the image gradient is high. The rationale is that high image gradient indicates probable object edges, and there the image values are between the object and the background gray levels. The threshold surface was determined by successive over-relaxation as the solution of the Laplace equation. This work proposes a different method to determine an adaptive threshold surface. In this new method, inspired by multiresolution approximation, the threshold surface is constructed with considerably lower computational complexity and is smooth, yielding faster image binarizations and better visual performance.

1 Introduction

Let us consider the problem of separating the objects from the background in a gray level image $I(x, y)$, where objects appear lighter (or darker) than the background. This can be done by constructing a threshold surface $T(x, y)$, and constructing the binarized image $B(x, y)$ by comparing the value of the image $I(x, y)$ with $T(x, y)$ at every pixel, via

$$B(x,y) = \begin{cases} 1 & \text{if } I(x,y) > T(x,y) \\ 0 & \text{if } I(x,y) \leq T(x,y). \end{cases} \quad (1)$$

It is clear that a fixed value of the threshold surface $T(x, y) = const.$ can not yield satisfactory binarization results for images obtained under nonuniform illumination and/or with a nonuniform background.

Yanowitz and Bruckstein in [1], motivated by the approach of Chow and Kaneko [2], proposed to construct a threshold surface by interpolating the image gray levels at the points where the image gradient is high. Indeed, high image gradients indicate probable object edges, where the image gray levels are between the object and the background levels. The threshold surface was required to interpolate the image gray levels at all the support points and to satisfy the Laplace equation at non-edge pixels. Such a surface was determined by a successive over-relaxation method (SOR) [1, 3].

Subsequent performance evaluation of several binarization methods showed that the Yanowitz-Bruckstein (YB) method was one of the best binarization methods [4]. However, the computational complexity of successive over-relaxation method is expensive: $O(N^3)$ for an $N \times N$ image, and the resulting binarization process is slow, especially for large images. Furthermore, the threshold surface tends to have sharp extrema at the support points, and this can degrade binarization performance.

We here follow the approach of YB and use image values at support points with high gradients to construct a threshold surface. However, we define a new threshold surface via a method inspired by multi resolution representations, like Laplacian Pyramids [5] or wavelets [6]. The new threshold surface is constructed as a sum of functions, formed by scaling and shifting of a predetermined function. This new threshold surface can be stored in two ways: either as an array of coefficients a_{ljk}, or as a conventional threshold surface $T(x, y)$, obtained as a sum of scaled and shifted versions of the given function, multiplied by appropriate coefficients a_{ljk}.

The threshold surface coefficients a_{ljk} are determined in $O(P \log(N))$ time, where P is the number of support points and N^2 is the image size. These coefficients can then be used to construct the threshold surface $T(x, y)$ over the entire image area N^2 in $O(N^2 \log(N))$ time or to construct the threshold surface over smaller region of the image of M^2 size in only $O(M^2 \log(N))$ time. Furthermore, the adaptive threshold surface can be made smooth over all the image domain.

This paper is organized as follows: Section 2 reviews the YB binarization method and the properties of threshold surfaces obtained by successive over-relaxation. Section 3 describes a new method, proposed to construct the threshold surface. Section 4 describes the implementation of the surface computation. Finally, Section 5 presents some experimental results, comparing the speed and binarization performance of the two methods.

2 The Yanowitz-Bruckstein Binarization Method

The essential steps of the YB binarization method method are the following:

1. Find the *support points* $\{p_i\}$ of the image $I(x,y)$, where the image gradient is higher then some threshold value G_{th},

$$\{p_i\} = \{(x_i, y_i) \big| |\nabla I(x_i, y_i)| > G_{th}\}. \qquad (2)$$

2. Find the threshold surface $T(x,y)$ that equals to the image values at the support points $\{p_i\}$ and satisfies the Laplace equation at the rest of the image points:

$$\begin{array}{l} T(p_i) = I(p_i) \\ \nabla^2 T(x,y) = 0 \quad \text{if } (x,y) \notin p_i. \end{array} \qquad (3)$$

   The solution of (3) is found by the SOR method.

3. Determine the binarized image $B(x,y)$ according to (1), *i.e.* by comparing I(x,y) with T(x,y).

The original method included also some pre- and post- processing steps (see Section 2 of [1]), omitted here for the sake of clarity.

The SOR starts with an approximate solution $t(x,y)$, and numerical iterations take it to the unique solution $T(x,y)$ of the Laplace equation. At each iteration, j, the Laplacian value of $t_j(x,y)$ is computed in each point (that should be zero for the exact solution), multiplied by some constant $1 \leq \lambda < 2$ and subtracted from the $t_j(x,y)$, to yield $t_{j+1}(x,y)$ as proposed by Southwell in [3]. Then, the values of $t_{j+1}(p_i)$ at the support points $\{p_i\}$ are reset to be equal to the image values at these points $I(p_i)$. Finally, the values of $t_{i+1}(x,y)$ at the boundary points are set to be equal to the values of their internal neighbors, thus implementing the Newmann boundary conditions. The iterative process can be described in semi-MatLab notation as follows:

$$\begin{array}{l} t_0(x,y) = I(x,y) \\ for \quad j = 1:N, \\ \quad t_j(x,y) = t_{j-1}(x,y) - \lambda \cdot \nabla^2 t_{j-1}(x,y) \\ \quad t_j(p_i) = I(p_i) \\ \quad t_j(1\ end, :) = t_j(2\ end-1, :) \\ \quad t_j(:, 1\ end) = t_j(:, 2\ end-1) \\ end \end{array} \qquad (4)$$

Each iteration requires $O(N^2)$ operations for N^2 grid points and there should be $O(N)$ iterations to converge to a solution, therefore, the method complexity is $O(N^3)$ [1]. The solution of (3) can be found in just a $O(N^2)$ time using multigrid methods [7, 8]. However it will become clear from the following paragraph that not only the speed of computation but also the properties of the threshold surface can be improved.

The general form of the solution of the (3) in the continuum limit is :

$$\phi(x,y) = \psi(x,y) - \sum_{i=1}^{P} q_i \cdot \log(\sqrt{(x-x_i)^2 + (y-y_i)^2}). \qquad (5)$$

Where $\psi(x,y)$ is a smooth and bounded function [9]. This solution has singularities at the support points. In the case of a problem discretized on a finite grid, the solution obtained by procedure (4) will be finite, yet it will have sharp extrema at the support points. These sharp extrema and especially the hanging 'valleys' between them can cause the unwanted 'ghost' objects on the binarized image. These ghost objects where eliminated by postprocessing step in[1], however, it is preferable to get rid of them already by a careful construction of the threshold surface. To illustrate sharp extremas at the support points and the hanging 'valleys' in between, Figure 1 shows a surface computed by SOR for 100 support points with random values in the range of $0..100$. The support points were randomly scattered over a 128×128 grid.

Ideally, a good threshold surface should indicate the local illumination level, therefore the threshold surface constructed by the successive over-relaxation is not optimal in this sense. The next section describes a new way to construct such a threshold surface.

3 The New Threshold Surface

We propose to construct and represent the threshold surface in the Multi Resolution framework, as a sum of functions, obtained by scaling and shifting of the single source function. Unlike Laplacian Pyramids [5, 10], where the coefficients are calculated on the basis of an original signal that is a priori known, in our case the complete threshold surface is unknown in advance, but only its approximate values at the support points: $T(p_i) = I(p_i)$. This section presents a way to construct surfaces that interpolate and approximate image values at the support points $I(p_i)$. First, a simple interpolation algorithm is presented. However, the interpolation surface obtained is discontinuous and can not serve as a good threshold surface. Therefore, a small modification to the interpolation algorithm is next presented, that results in a continuous and smooth approximation surface.

Let us consider a unit step source function, given by

$$G_{000}(x,y) = \begin{cases} 1 & \text{if } (x,y) \in \Omega(I) \\ 0 & \text{if } (x,y) \notin \Omega(I). \end{cases} \qquad (6)$$

Here $\Omega(I)$ denotes the set of all the image points (x,y) and p_i denotes the i-th support point. All the other functions

we shall use are generated by downscaling of this source function and shifting the downscaled functions to various positions in the image plane

$$G_{ljk}(x,y) = G_{000}(x \cdot 2^l - j, y \cdot 2^l - k), \qquad (7)$$

where $l = 0, \ldots, \log_2(N)$ is a scale factor and $j,k \in \{0, \ldots, 2^l - 1\}$ are spatial shifts.

The threshold surface will be given by

$$T(x,y) = \sum_{l=0}^{\log_2(N)} \sum_{j,k=0}^{2^l-1} a_{ljk} G_{ljk}(x,y). \qquad (8)$$

3.1 Interpolation Algorithm

Let us introduce an algorithm to calculate the "decomposition coefficients" a_{ijk} in order to obtain an interpolating surface $T(x,y)$ given as (8), that passes exactly through all the support points $T(p_i) = I(p_i)$.

The algorithm runs as follows:

1. The decomposition coefficient a_{000} is set equal to the average of all the support points,

$$a_{000} = \langle I(p_i) \rangle = \frac{1}{P_{000}} \sum_{i=1}^{P_{000}} I(p_i). \qquad (9)$$

After Step 1, every support point $p_i^{(0)}$ is already approximated by the average a_{000}, so it remains only to interpolate the difference between the value of every support point and the average.

2. The values of the support points are updated as follows:

$$p_i^{(1)} = p_i^{(0)} - a_{000}. \qquad (10)$$

3. The image is divided into four "quadtree" cells, with corresponding indexes $\{jk\}$ relating to the spatial position of the cell: $\{00, 01, 10, 11\}$. The average of the updated support points $p_i^{(1)}$ of each cell jk is calculated to yield the appropriate decomposition coefficient a_{1jk}:

$$a_{1jk} = \frac{1}{P_{1jk}} \sum_{p_i^{(1)} \in S_{1jk}} p_i^{(1)}. \qquad (11)$$

Here $p_i^{(1)} \in S_{1jk}$ denotes a support point p_i that belongs to the cell at the 1-st resolution level, located at the (j,k) spatial position. P_{1jk} denotes the number of support points in this cell.

4. After Step 3, the values of support points in each cell jk are approximated by $a_{000} + a_{1jk}$, so their values are updated to be:

$$p_i^{(2)} = p_i - a_{000} - a_{1jk} = p_i^{(1)} - a_{1jk}. \qquad (12)$$

5. Steps 3 and 4 are repeated for successive resolution levels. At every resolution level $(l-1)$ each of the 4^{l-1} cells of this level is divided into four cells to yield 4^l cells at the resolution level l. The coefficients a_{ljk} of the cells at level l at the (j,k) spatial position are set to be equal to the average of the residual values of the support points, belonging to this cell:

$$a_{ljk} = \frac{1}{P_{ljk}} \sum_{i \in S_{ljk}} p_i^{(l)}. \qquad (13)$$

Here $p_i^{(l)} \in S_{ljk}$ denotes a support point p_i that belongs to the cell at level l, located at the (j,k) spatial position. P_{ljk} denotes the number of support points in this cell. After calculation of the coefficients a_{ljk}, the values of the support points are updated by,

$$p_i^{(l+1)} = p_i^{(l)} - a_{ljk}. \qquad (14)$$

6. The procedure ends at the highest resolution level L ($L = \log_2(N)$), when the size of the cell equals to one pixel. At this step there is at most one support point in every cell jk, with a residual value $p_i^{(L)}$. The coefficient a_{Ljk} is set to $a_{Ljk} = p_i^{(L)}$.

The threshold surface, constructed in accordance with equations (6-8) with the coefficients a_{ljk} obtained by the algorithm as described in steps **1-6**, will be an interpolation surface of the support points $p_i^{(0)}$, i.e. it will pass through every support point. This can be proved by the following argument:

Consider some arbitrary support point $p_i^{(0)}$. The value of the threshold surface at this point will be

$$T(p_i) = \sum_{l=0}^{L} a_{lj_lk_l}. \qquad (15)$$

Where, the j_lk_l chooses at every level l the cell that contains the p_i.

On the other hand, the residual value $p_i^{(L+1)}$ of the support point p_i equals to (Step 6):

$$p_i^{L+1} = p_i^{(0)} - a_{000} - a_{1j_1k_1} - \ldots - a_{Lj_Lk_L} = 0, \qquad (16)$$

which can be rewritten as

$$I(p_i) \equiv p_i^{(0)} = a_{000} + a_{1j_1k_1} + \ldots + a_{Lj_Lk_L}. \qquad (17)$$

From (15) and (17) it follows that for an arbitrary support point p_i, $T(p_i) = I(p_i)$.

Figure 2 shows the interpolation surface, obtained by our method for the same set of support points that was used for the over-relaxation solution shown in Figure 1.

3.2 Approximating Source Function

The method presented in the previous section yields a surface that interpolates the support points. However, the resulting interpolation surface is discontinuous. In order to obtain an n-times continuously differentiable approximation surface, the source function (6) must be substituted by n-times continuously differentiable function vanishing together with n first derivatives at the boundary of its support.

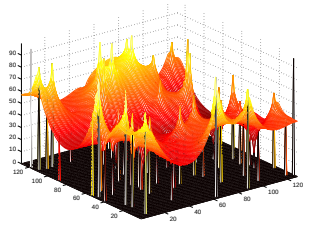

Figure 1: Solution by the Over-relaxation method

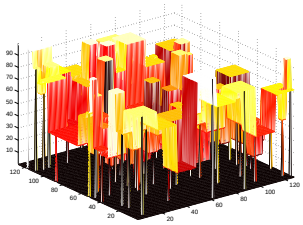

Figure 2: Interpolating surface, obtained by the new interpolation method.

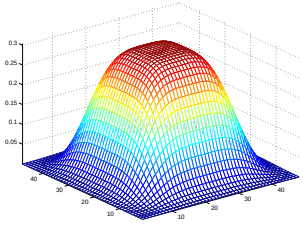

Figure 3: The source function, given by (18)

.

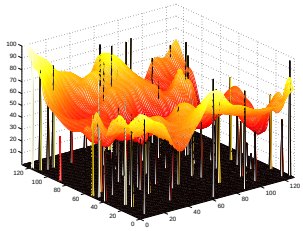

Figure 4: An approximating surface, obtained with source function (18)

In the practical case of finite grid it is enough to consider a source function having a value and derivatives small enough at the boundary. However, there are three additional requirements from the source function.

- Approximation: It should have value close to 1 in the domain of its cell.
- Normalization: The integral of the source function over its support must be equal to the image area.
- Smoothness: It should decrease gracefully towards the boundary of its support.

The first two requirements are necessary in order to build the threshold surface approximating the support points and the third one in order to have it practically smooth.

There are infinitely many possible source functions, satisfying these requirements. After some experimentation with several simple functions, we have chosen a source function with support $[-1,2]\times[-1,2]$, extending the image area $[0,1]\times[0,1]$. Therefore, the threshold surface (8) is constructed with scaled functions, overlapping at each resolution level. The new source function is given by

$$G_{000}(x,y)=\begin{cases}\dfrac{e^{-(x-\frac{1}{2})^4-(y-\frac{1}{2})^4}}{\int_{-1}^{2}\int_{-1}^{2} e^{-(x-\frac{1}{2})^4-(y-\frac{1}{2})^4}}, & \text{if}\{x,y\}\in[-1,2]^2\\ 0 & \text{if}\{x,y\}\notin[-1,2]^2.\end{cases} \quad (18)$$

The point $\{x,y\}=\{\frac{1}{2},\frac{1}{2}\}$ is the center of the image, spanning over $[0,1]\times[0,1]$. Figure 3 shows the source function (18). The support points that determine the decomposition coefficients lie in the central cell $[0,1]\times[0,1]$, where the source function (18) is practically flat. Eight periphery cells will overlap neighboring functions thus making the threshold surface smooth.

Figure 4 shows the smooth threshold surface, constructed with the source function (18) for the same set of support points that was used to construct the interpolated surfaces of Figures 2 and 1. This figure (as well as figures 1 and 2) shows also the support points by vertical spikes. Some of the support points of Figure 4 are lying far from the threshold surface. This is due to the fact that support points are taken to have random values for demonstration purposes

a_{000}	a_{100}	a_{101}	...
P_{000}	P_{100}	P_{101}	...

Table 1: Array *coeffs*. Contains decomposition coefficients a_{ljk} and number of support points P_{ljk} in the cell ljk.

and therefore the approximating surface passing between them is far from the support points in some places. In the real cases, the neighboring support points usually have similar values and approximation surface pass closer to them. The new threshold surface is smooth. It does not necessarily pass exactly through the support, points, however this is an advantage rather than disadvantage, because if several neighboring support points have substantially different and 'noisy' values this indicates either that the threshold surface is under-sampled by the support points or that there is some error or noise in their values. In both cases there is not enough information at the support points about the threshold surface and probably the best solution is to set the threshold surface somewhere in between, like in the proposed approximation algorithm.

4 Implementation

4.1 Data Structures

The basic data structures are two arrays:

The first array is called *coeffs* (Table 1). It stores the decomposition coefficients of the cells a_{ijk} in the first row and the number of support points P_{ljk} of the corresponding cell in the second row. a_{ljk} denotes the decomposition coefficient of the cell ljk, which is located at the (j,k) spatial position, at level l of the resolution. P_{ljk} stores the number of support points in this cell. First column of *coeffs* stores the single coefficient of the lowest level a_{000} and the total number of support points $P\equiv P_{000}$. Following are four columns of coefficients of the first level $(a_{100},\ldots,a_{111})$ and number of points in each of these cells $(P_{100},\ldots,P_{111})$, etc.

Every support point belongs to one and only one cell lj_lk_l at every resolution level l. There are $\lg(N)$ different

p_1	p_2	$\cdots$	p_p
$\cdots$	$\cdots$	$\cdots$	$\cdots$
$p_{log_2(N)1}$	$p_{log_2(N)2}$	$\cdots$	$p_{log_2(N)p}$

Table 2: Array *pointarr*. Column i contains the indices of the cells containing p_i.

Grid	32×32	64×64	128×128	256×256
SOR	0.39	2.35	18.9	160.7
FA	0.1 (0.01)	0.4 (0.01)	1.55 (0.06)	7.45 (0.27)

Table 3: Runtimes of SOR and FA.

resolution levels, starting from single cell of size $N \times N$ at level 0 to the N^2 cells of size 1×1 at level $\lg(N)$.

The second array, called *pointarr* (Table 2), has P columns and $1 + \lg(N)$ rows. Every column of *pointarr* contains the current value of the support point $p_i^{(l)}$ in the first row, and the indexes ind_{il} in other rows. These indexes refer to the cells which contain p_i at every level l: *coeffs*$[:, ind_{il}] = [a_{ljk}; P_{ljk}]$. Figure 5 shows an example of a point, which belongs to $cell_{000}$ at level 0 (as every point does), $cell_{100}$ at level 1 , to cell $cell_{211}$ at level 2 etc.. This point contributes in the construction of the threshold function only through the coefficients $a_{000}, a_{100}, a_{211}, \ldots$. These coefficients are stored in the first row, columns $1, 2, 11, \ldots$ of array *coeffs* (Table 1). Thereby, the column of *pointsarr*, corresponding to this point will have values $1, 2, 11, \ldots$ in its second, third, fourth ... rows.

4.2 Algorithm description

1. Array *pointarr* (Table 2) is created and gradually filled. Every column i of this table contains value of the point p_i in the first raw. For every point p_i a cell $lj_{il}k_{il}$ that contains it at each level l l=0, ..., $\lg(N)$ is determined . The positions of these cells in the array *coeffs* (Table 1) are filled into rows $2, \ldots, N$, of the i-th column of *pointarr*, and simultaneously, for every encountered cell the counter of the points belonging to this cell is increased in the array *coeffs*. This requires $\lg(N)$ calculations of the cell index and $\lg(N)$ increments of the point counters for each of the P support points entered into the $\lg(N)$ cells.

2. The coefficients a_{ljk} in the array *coeffs* are calculated. a_{000} is set to be an average value of all the points (9). Next, the value of every point in *points* is updated: the average value is subtracted from it (10).

3. Step 2 is repeated for a higher level l.

4. The threshold surface is built based on the *coeffs* and the basis function (18). This requires $O(N^2 \lg(N))$ operations. So, an approximation surface for P support points scattered over N^2 grid points is determined as a set of coefficients using $O(P \lg(N))$ operations and constructed explicitly, using $O(N^2 \lg(N))$ operations.

5 Experimental results

Speed comparison:

The two methods, YB with adaptive threshold surface obtained by SOR and the new one with adaptive threshold surface obtained by multiresolution approximation were compared for speed and quality of binarization. The programs were implemented in MATLAB 5.3 and ran on an IBM-Thinkpad-570 platform with 128MB RAM and a Pentium-II 366 MHz processor.

Table 3 presents the speed comparison results for the two methods. The test images are successively increasing portions of one of the images. The support points in all cases constituted the 1% of the image points with the highest gradients. The run times are given in seconds. For the new threshold surface approximation method, referred to as *Fast Approximation* or *FA*, two runtimes are given. The first one is the runtime for the full image binarization with the explicit threshold surface $T(x, y)$ and it should be compared with the runtime of the SOR method. The second is the considerably shorter runtime necessary to obtain the decomposition coefficients a_{ijk} that implicitly contain all the information about the threshold surface and can then be used for an efficient 'region-of-interest' processing.

Binarization Performance:

The binarization methods were tested on several of the images from the 'images' toolbox of Matlab. A smooth parabolic function was added to the image of 'IC' in order to simulate a nonuniform illumination. About a half of the tested images showed similar visual performance of the two methods, while another half evidenced the advantage of FA. Some of the typical results, evidencing the advantage of FA are presented here.

Figures from 6 to 14 show 3 images and 6 corresponding binarization results. The size of all the images is 256×256. The Figures show an original image followed by two figures, showing images binarized with the threshold surfaces constructed by SOR and then by FA. The binarization processing time for images with the SOR threshold surface varied between 161.1 and 162.7 sec, while for images obtained with FA varied between 7.2 to 7.8 seconds. Obviously, the speed advantage of the Fast Approximation will be greater for larger images.

The image 'IC' is a good example of 'ghost' objects that appear as white areas between the conductor lines in the image binarized with the SOR, Figure 13. They are almost absent in the image binarized with the new threshold surface and shown on Figure 14.

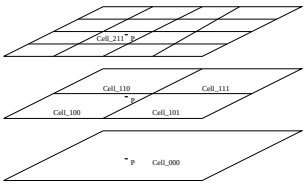

Figure 5: Cell Hierarchy.

Figure 6: Barbara.

Figure 7: Binarization of Barbara with SOR.

Figure 8: Binarization of Barbara with FA.

Figure 9: Trees.

Figure 10: Binarization of Trees with SOR.

Figure 11: Binarization of Trees with FA.

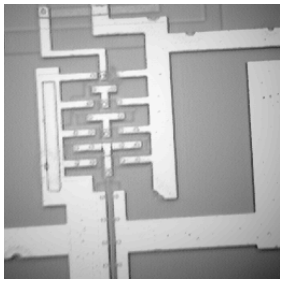

Figure 12: IC.

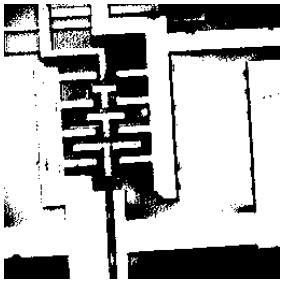

Figure 13: Binarization of IC with SOR.

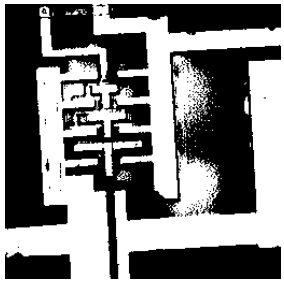

Figure 14: Binarization of IC with FA.

Acknowledgements

We would like to thank Danny Barash from HP Labs Israel, for his valuable comments.

6 Concluding Remarks

This work proposes efficient procedures for gray level image binarization motivated by the method proposed in [1]. The new threshold surface is constructed in the framework of multi-resolution analysis, with a considerably lower computational complexity and hence in a much shorter time even for small images. The new threshold surface can be made smooth and by the nature of its construction should be similar to the local illumination level. These qualities allowed us to often obtain better visual performance of the binarization process. In particular, the quantity and size of 'ghost' objects in some images was lower.

References

[1] S.D.Yanowitz and A.M.Bruckstein. A new method for image segmentation. *Computer Vision, Graphics and Image Processing*, 46:82–95, 1989.

[2] C.K.CHow and T.Kaneko. Automatic boundary detection of the left-ventricle from cineangiograms. *Comput. Biomed.*, 5:388–410, 1972.

[3] V.R.Southwell. Relaxation methods in theoretical physics. *Oxford University Press*, 1946.

[4] D.Trier and T.Taxt. Evaluation of binarizaiton methods for document images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:312–315, 1995.

[5] P.J.Burt and E.H.Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on Communications*, 31(4):532–540, 1983.

[6] Stephane G.Mallat. *A Wavelet Tour of Signal Processing*. Academic Press, 1999, pp.221-224.

[7] W. L. Briggs. *A Multigrid Tutorial*. Philadelphia, PA:SIAM, 1987.

[8] R.Kimmel and I.Yavneh. An algebraic multigrid approach for image analysis. *SIAM Journal on Scientific Computing*, to be submitted, 2001.

[9] R.Courant and D.Hilbert. *Methods of Mathematical Physics*. Interscience Publishers, 1953.

[10] J.L.Crowley and R.M.Stern. Fast computation of the difference of low-pass transform. *IEEE Transactions on PAMI*, 6(2), 1984.