A Shape Preserving and Noise Removing Image Enhancer through Regularization

George P. Choung

Department of Mathematics
North Carolina State University

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	12.Feb.01	MAJOR REPORT

4. TITLE AND SUBTITLE
A SHAPE PRESERVING AND NOISE REMOVING IMAGE ENHANCER THROUGH REGULARIZATON

5. FUNDING NUMBERS

6. AUTHOR(S)
2D LT CHOUNG GEORGE P

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NORTH CAROLINA STATE UNIVERSITY

8. PERFORMING ORGANIZATION REPORT NUMBER
CI01-45

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
THE DEPARTMENT OF THE AIR FORCE
AFIT/CIA, BLDG 125
2950 P STREET
WPAFB OH 45433

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Unlimited distribution
In Accordance With AFI 35-205/AFIT Sup 1

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

14. SUBJECT TERMS			15. NUMBER OF PAGES 27
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Abstract

This project concerns the development of an image reconstruction algorithm which removes noise from a corrupted image and recovers patterns and textures. The textures this algorithm detects are edges and smooth, gray scale regions. In previous attempts of this kind, images were recovered with nice edges, but blockiness remained in the smooth regions. Our goal is to recover both types of image composition. We do this by a least squares minimization method and a regularization process. The regularization is the key element that allows us to differentiate the edge and the smooth regions. The computation generated by the algorithm tends to create a large system to solve. We will use an algebraic multigrid solver to minimize the time for solving these large systems.

Introduction

Signal processing and specifically image restoration span a wide variety of uses. Whether the image is an old photograph, an infrared scan, or a satellite image, it is clear that noise filtering and shape recovery are two important aspects of creating a clear picture.

The first question we must answer is what is the need for this type of image recovery. Take a satellite image for example. Due to many different reasons, a satellite image may be compromised by noise. Whether this noise is physical as in partial cloud cover, patterned as in noise added during the transmission of the signal, or random due to defects in equipment, we must try to overcome these different sources of noise. But decreasing noise is not the only task. Preserving the textures, called shape preservation, is equally important. A noiseless picture is not much benefit unless the image you are observing is an accurate representation of what is being observed.

In developing an algorithm, it is beneficial to look at the one-dimensional case. Consider a one-dimensional image, I(x), as in Fig 1. Notice the three different types of

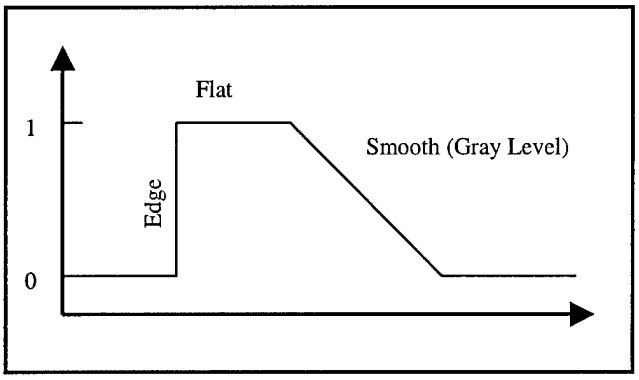

Fig. 1

texture regions that compose this picture. There are three flat regions: two with a level of zero, and one with a level of 1. The vertical line represents an edge, while the sloped line

segment represents a smooth region. A typical noisy observation of I(x) might occur as what is seen in Fig 2. Depending on the amount of noise in the image, the human eye

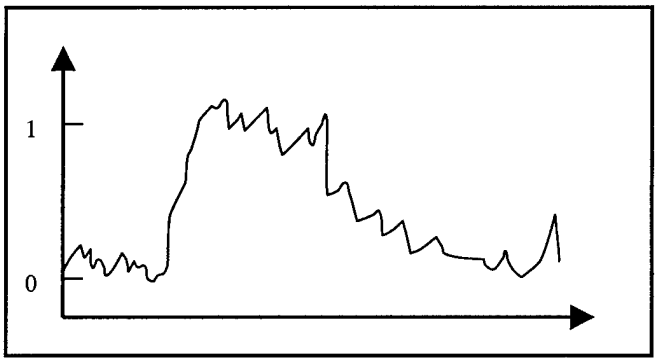

Fig. 2

itself cannot perceive the desired information. That is why we need a method to recover the image. A typical reconstructed image is shown in Fig 3.

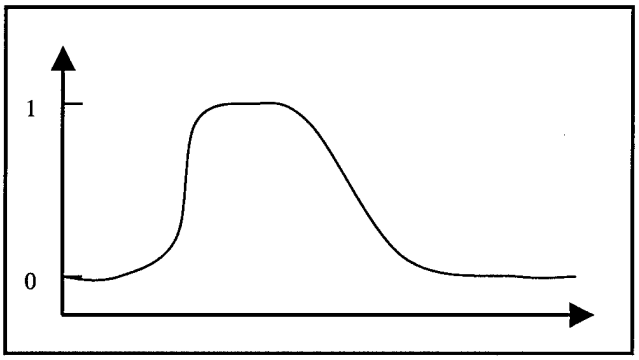

Fig. 3

Notice that the difference between a reconstructed edge and a smooth region is minor. A good algorithm will detect the difference and display them accordingly. This so-called shape preserving is the other aspect of image recovery that is important.

As you can see there are two different aspects to the noisy image. One is the high frequency noise that the noise created. The other is a low frequency effect that represents the actual shape of the image. First we describe the reconstruction though the use of partial differential equations. We apply the heat equation, which is well known to have

smoothing properties: solving the heat equation with a discontinuous initial condition will yield a

$$\frac{\partial u}{\partial t} = \sigma^2 \frac{\partial^2 u}{\partial x^2} \qquad [1]$$
$$u(0,x) = y(x)$$

continuous and smooth solution. This smoothing however, tends to smear the edges.

In order to retain and recover the edges, it has been proposed that replacing the heat equation with the non-linear diffusion equation.

$$\frac{\partial u}{\partial t} = \sigma^2 |u_x| \frac{\partial}{\partial x}\left(\frac{\frac{\partial u}{\partial x}}{|u_x|}\right) \qquad [2]$$

The two dimensional version of this nonlinear diffusion equation is known as the mean curvature flow. The nonlinear diffusion thus provides smoothing in the direction of the tangent, rather than in the normal direction. Thus, the edges are not smeared by this procedure.

Now we describe an alternative approach based on a variational formulation. We consider the following minimization

$$J(u) = \frac{1}{2}\int |u(x) - y(x)|^2 dx + \frac{\beta}{2}\int |u_x|^2 dx . \qquad [3]$$

The first term of the cost functional, J, is the least squares fit, u(x), to the observed image, y(x). Minimizing the norm of the difference between the solution and the observed image insures that the solution stays relatively close to the observed image. This is an important term, however if it were the only term, the minimum solution would always be the observed data. The second term is called Gaussian Regularization. This regularization term is to control the total variation of the reconstructed image, so the

oscillation of the constructed image is controlled. The necessary and sufficient optimality condition is given by

$$-\beta u_{xx} + u - y = 0. \qquad [4]$$

This is the steady state of the forced, time-dependent heat equation

$$\frac{\partial u}{\partial t} = \beta \frac{\partial^2 u}{\partial x^2} + (y - u). \qquad [5]$$

In order to reconstruct the image with edges we consider a regularization with a bounded variation of the image, u. We consider instead of Eq. 3, the cost functional

$$J(u) = \frac{1}{2}\int |u(x) - y(x)|^2 dx + \beta \int |u_x| dx \qquad [6]$$

It will be shown that the optimality condition of this problem is written as

$$-\beta\left(\frac{u_x}{|u_x|}\right)_x + u = y \qquad [7]$$

Then we can observe that the nonlinear diffusion term is similar to the one that appears in the method mentioned above. Now it is well known that this BV regularization method works well for images with edges, but our objective is to introduce a new class of regularization to avoid the blockiness found in the BV regularization method.

Regularization Method

In this section, we will describe our idea and approach using the one-dimensional case. This approach can then be extended to multiple dimensions. The regularization method we employed is based on bounded variations of the image. Here our domain, Ω, is the interval, (0,1). The function we would like to minimize is

$$J(u) = \frac{1}{2}\int_0^1 |u(x) - z(x)|^2 dx + \beta_1 BV(u) + \beta_2 BV(u_x), \qquad [8]$$

where $u \in H_0^1(0,1)$. $BV(u)$ denotes the bounded variation semi-norm of u. If u is absolutely continuous, then

$$BV(u) = \int_0^1 |u_x| dx. \qquad [9]$$

Thus in the rest of our discussion, we will continue to use the right hand side of the previous equation as our notation for BV(u). We use a bounded variational approach since this allows for jump discontinuities, which will be important later on in our discussion. This first regularization term, BV(u), represents a total variation of the image and helps to restore regions with edges. In other words, in a one-dimensional image, an image that is a piece-wise constant would be readily restored.

Now while this term alone would work well for an image with such block characteristics, it does poorly in restoring what would be represented in a one-dimensional image as a slope. Smooth slopes will be restored to a series of jagged stair steps. Basically it will try to give edge-like characteristics to the smooth region. In a gray scale image, the gray would be restored to blocky regions of black, white, and gray levels if we employ only the first BV term. To restore the intermediate gray scale, we

introduce the second BV term. This is a bounded variation on the derivative, u_x. If u_x is absolutely continuous, then

$$BV(u_x) = \sum_{i=1}^{m} \left[\int_{s_i}^{s_{i+1}} |u_{xx}| dx + |[u_x](s_i)| \right] \qquad [10]$$

where

$[u_x](s_i) = u_x(s_i^+) - u_x(s_i^-)$ is the jump discontinuity of the derivative at $x = s_i$. This BV term is the shape preserving entity, which we need to prevent blockiness.

Now these two terms can be used to relate the effects on each other on the way the image is restored. The importance of the relative weight factors, β_1 and β_2, can be seen by letting β_2 be zero thereby taking away gray scale resolution. On the other hand letting β_1 equal to zero would make edges tend to disappear. The goal is to find a balance between these two extremes to find a way to reproduce both types of image composition. For example we can take a piecewise linear function on the interval (0,1) that has finite restoration energy

$$Q(u) = \beta_1 BV(u) + \beta_2 BV(u) = \beta_1 \sum_{i=1}^{m} |\lambda_i| (s_{i+1} - s_i) + \beta_2 \sum_{i=1}^{m-1} |\lambda_{+1i} - \lambda_i|, \qquad [11]$$

where u is linear on each interval (s_i, s_{i+1}) with slope λ_i.

In the case of a discontinuous image we let ϕ be a piecewise constant function, where $\phi = 0$ on $(0, \frac{1}{2})$ and $\phi = 1$ on $(\frac{1}{2}, 1)$ can be approximated by the piecewise linear, continuous function

$$\phi_\delta = \begin{cases} 0 & x \in \left(0, \dfrac{1-\delta}{2}\right) \\ \dfrac{1}{\delta}\left(x - \dfrac{1}{2}\right) + \dfrac{1}{2} & x \in \left(\dfrac{1-\delta}{2}, \dfrac{1+\delta}{2}\right) \\ 1 & x \in \left(\dfrac{1+\delta}{2}, 1\right) \end{cases}. \qquad [12]$$

This allows us to bound the variation of u_x, for otherwise we would have $BV(u_x) = \infty$.

Now we have

$$Q(\phi_\delta) = \beta_1 + \frac{2\beta_2}{\delta} \qquad [13]$$

Ideally we would like the restoration energy to have equal contributions from the two bounded variation terms. This leads to a proportionality relationship between $\beta_1\delta$ and β_2. Since $BV(\varphi_\delta) = BV(\varphi)$ and $|\phi_\delta - \phi|^2 = \frac{\delta}{3}$, and if we take $\beta_2 \propto \delta^2$ then we have

$$Q(\phi_\delta) \le C|\phi_\delta - \phi|^2 \qquad [14]$$

for some constant C. That is, the three terms in the cost functional are balanced. This provides us with an idea of values we can use for the parameters, β_1 and β_2; namely that the three terms in the cost functional must be roughly proportional to each other.

Mathematical Formulation

Now we turn again to the variational problem. We continue in the one-dimensional case as before. We state the following theorem.

Theorem 1: There exists a u that minimizes J.

Proof: We define a normed space, X, as the space of all continuous functions on (0,1) with bounded variations, $BV(u) + BV(u_x) < \infty$. X is equipped with the norm

$$|u|_X = \int_0^1 |u| dx + BV(u) + BV(u_x). \qquad [15]$$

Then it can be shown that X is complete, that is X is a Banach Space. Suppose $\{u_n\}$ is a minimizing sequence of J. That is, such that $\lim_{n\to\infty} J(u_n) = \inf J(u)$ over $L^1(0,1)$. Now there exists a bound for the subsequence. Since X is compactly embedded in $L^2(0,1)$, there must exist a subsequence $\{ u_{\hat{n}} \}$ of $\{u_n\}$ that converges strongly in $L^2(0,1)$ and weakly star in X to u. Since the norm is weakly star, lower semi-continuous, we have

$$J(u) \le \liminf_{\hat{n}\to\infty} J(u_{\hat{n}}). \qquad [16]$$

Therefore, we have a u that minimizes the cost functional, J.

We turn the discussion to the necessary and sufficient condition for optimality. To do this first let u be the minimizer for the cost functional J. Then for all $0 < t < 1$, we have $J(u + t(v-u)) \ge J(u)$ for all v. Now we have

$$\begin{aligned} J(u+t(v-u)) - J(u) = \int_0^1 t(u-z)(v-u) + \frac{t^2}{2}(v-u)^2 dx + \beta_1 \int_0^1 |u_x + t(v-u)_x| - |u_x| dx \\ + \beta_2 \int_0^1 |u_{xx} + t(v-u)_{xx}| - |u_{xx}| dx \end{aligned} \qquad [17]$$

Since we know that for A, B $\in$ R,

$$|A+t(B-A)| = |(1-t)A+tB| \le (1-t)|A|+t|B|$$
$$|A+t(B-A)|-|A| \le t(|B|-|A|) \qquad [18]$$

Using this fact, coupled with the expressing for J above, we have

$$0 \le J(u+t(v-u))-J(u) \le \int_0^1 t(u-z)(v-u)+\frac{t^2}{2}(v-u)^2\,dx + t\beta_1\int_0^1 |v_x|-|u_x|dx + t\beta_2\int_0^1 |v_{xx}|-|u_{xx}|dx \qquad [19]$$

Now we have the one-dimensional case for optimality by letting $t \to 0$:

$$(u-z, v-u)+\beta_1(|v_x|-|u_x|)+\beta_2(|v_{xx}|-|u_{xx}|) \ge 0 \quad \text{for all } v \in L^1(0,1). \qquad [20]$$

Now let $W = \{H^2(0,1)\cap H_0^1(0,1);\ u\in L^2(0,1);\ u_x, u_{xx}\in L^2(0,1);\ u(0)=u(1)=0\}$.

Then if we assume that the minimizer, u(x), is an element of W. Then we can derive the differential form of the necessary and sufficient condition for optimality. First we note that $J(u+th) \ge J(u)$ for all $h \in W$ and $t \in R$. In order to write the optimality condition in an equation form, we consider for $\varepsilon \ge 0$,

$$J_\varepsilon(u) = \frac{1}{2}\int_0^1 |u-z|^2\,dx + \beta_1\int_0^1 \sqrt{\varepsilon^2+|u_x|^2} + \beta_2\int_0^1 \sqrt{\varepsilon^2+|u_{xx}|^2}\ . \qquad [21]$$

As we will see in later discussion, adding the ε avoids the difficulty of dealing with singularities like $\frac{1}{|u_x|}$, where $u_x = 0$. Now doing the following calculation

$$\begin{aligned}
&\sqrt{\varepsilon^2+|(u+th)_x|^2}-\sqrt{\varepsilon^2+|u_x|^2} \\
&= \frac{|(u+th)_x|^2-|u_x|^2}{\sqrt{\varepsilon^2+|(u+th)_x|^2}+\sqrt{\varepsilon^2+|u_x|^2}} \\
&= \frac{2tu_xh_x+t^2|h_x|^2}{\sqrt{\varepsilon^2+|(u+th)_x|^2}+\sqrt{\varepsilon^2+|u_x|^2}}
\end{aligned} \qquad [22]$$

and similarly

$$\begin{aligned}
&\sqrt{\varepsilon^2+\left|(u+th)_{xx}\right|^2}-\sqrt{\varepsilon^2+\left|u_{xx}\right|^2} \\
&=\frac{\left|(u+th)_{xx}\right|^2-\left|u_{xx}\right|^2}{\sqrt{\varepsilon^2+\left|(u+th)_{xx}\right|^2}+\sqrt{\varepsilon^2+\left|u_{xx}\right|^2}}. \\
&=\frac{2tu_{xx}h_{xx}+t^2\left|h_{xx}\right|^2}{\sqrt{\varepsilon^2+\left|(u+th)_{xx}\right|^2}+\sqrt{\varepsilon^2+\left|u_{xx}\right|^2}}
\end{aligned} \qquad [23]$$

Applying these two formulas to the optimality condition and forming a difference quotient, we are left with

$$\begin{aligned}
\frac{J(u+th)-J(u)}{t} \geq \int_0^1 (u-z)h+\frac{t}{2}|h|^2\,dx+\beta_1\int_0^1 \frac{2u_xh_x+t\left|h_x\right|^2}{\sqrt{\varepsilon^2+\left|(u+th)_x\right|^2}+\sqrt{\varepsilon^2+\left|u_x\right|^2}}dx \\
+\beta_2\int_0^1 \frac{2u_{xx}h_{xx}+t\left|h_{xx}\right|^2}{\sqrt{\varepsilon^2+\left|(u+th)_{xx}\right|^2}+\sqrt{\varepsilon^2+\left|u_{xx}\right|^2}}dx
\end{aligned} \qquad [24]$$

for all h ∈ W and t ∈ R.

Now taking the limit of this expression as t approaches 0, we have

$$\int_0^1 (u-z)h+\beta_1\frac{u_xh_x}{\sqrt{\varepsilon^2+\left|u_x\right|^2}}+\beta_2\frac{u_{xx}h_{xx}}{\sqrt{\varepsilon^2+\left|u_{xx}\right|^2}}dx=0, \qquad [25]$$

for all h ∈ W. This is the weak variational form. This optimality condition can also be written in the differential form as

$$(u-z)-\beta_1\frac{\partial}{\partial x}\left(\frac{u_x}{\sqrt{\varepsilon^2+\left|u_x\right|^2}}\right)+\beta_2\frac{\partial^2}{\partial x^2}\left(\frac{u_{xx}}{\sqrt{\varepsilon^2+\left|u_{xx}\right|^2}}\right)=0, \qquad [26]$$

with $u(0) = u(1)$ and $u_{xx}(0) = u_{xx}(1) = 0$, for $\varepsilon \geq 0$.

Numerical Algorithm

Now that we have the mathematics behind the approach to the solution we must develop a numerical algorithm that can be readily implemented. Using the formula we derived for the one-dimensional case, we can obtain the two-dimensional form in $\Omega = (0,1)\times(0,1)$ and add a strategically placed $\varepsilon > 0$. Then the formula

$$(u - z, \phi) + \beta_1 \left(\frac{\nabla u}{\sqrt{\varepsilon^2 + |\nabla u|^2}}, \nabla \phi \right) + \beta_2 \left(\frac{\Delta u}{\sqrt{\varepsilon^2 + |\Delta u|^2}}, \Delta \phi \right) = 0 \qquad [27]$$

is in the differential form where $\nabla u = (u_x, u_y)$, the gradient, and $\Delta u = u_{xx} + u_{yy}$, the Laplacian. In fact, if we define $\varphi(s) = \sqrt{\varepsilon^2 + s}$, where $s \geq 0$, then it can be shown that it is the optimality condition for the minimization of

$$J_\varepsilon(u) = \frac{1}{2} \int_0^1 \left[|u - z|^2 + \varphi\left(|\nabla u|^2\right) + \varphi\left(|\Delta u|^2\right) \right] dx . \qquad [28]$$

We propose that the fixed point iterate is

$$u^{k+1} - \beta_1 \nabla \left(\frac{\nabla u^{k+1}}{\sqrt{\varepsilon^2 + |\nabla u^k|^2}}, \phi_x \right) + \beta_2 \Delta \left(\frac{\Delta u^{k+1}}{\sqrt{\varepsilon^2 + |\Delta u^k|^2}}, \phi_{xx} \right) + \mu \Delta\Delta (u^{k+1} - u_k) = 0 , \qquad [29]$$

where μ is arbitrarily small and is used to insure that u^k is in the $H^2(\Omega)$.

We state the following theorem of the convergence properties of the proposed algorithm.

Theorem 2: The cost, $J_\varepsilon(u^k)$, is monotonically decreasing as the iteration progresses. That is to say that $J_\varepsilon(u^{k+1}) \leq J_\varepsilon(u^k)$ for all k. The next part of the theorem states that $\lim_{k\to\infty} J_\varepsilon(u^k) = \inf_{u \in H^2(0,1) \cap H_0^1(0,1)} J_\varepsilon(u)$. Finally suppose that u^k is bounded in $H^2(\Omega)$. Then

u^k converges weakly to the unique solution u in $H^2(\Omega)$. That is to say that for all $\phi \in H^2(0,1) \cap H_0^1(0,1)$, we have the equation

$$\beta_1\left(\varphi'\left(|\nabla u|^2\right)\nabla u, \nabla\phi\right) + \beta_2\left(\varphi'\left(|\nabla u|^2\right)\Delta u, \Delta\phi\right) + (u - z, \phi) = 0 . \qquad [30]$$

Implementation of Algorithm

In our implementation, we will take the proposed algorithm and create the discretized formulation. First, we establish that the domain is the unit square, $(0,1)\times(0,1)$ in R^2. We create a uniform n by n size mesh of the domain thereby indicating n^2 pixels. Now letting $h=\frac{1}{n}$, and creating index variables i and j such that $0 \le$ i, j ≤ 1. Then the solution for u at (i, j), as approximated by $u_{i,j}$, is at the mid-point $x_{i,j}$, where $x_{i,j}=\left(\left(i-\frac{1}{2}\right)h,\left(j-\frac{1}{2}\right)h\right)$.

Now we represent the Laplacian, -Δ, in terms of the central difference scheme and we obtain the matrix

$$S_h = I \otimes H_0 + I \otimes H_0, \qquad [31]$$

where ⊗ is the Kronecker product and H_0 is the n by n tri-diagonal matrix represented as

$$H_0 = n^2\begin{bmatrix} 1 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & \ddots & \ddots & \\ & & \ddots & 2 & -1 \\ & & & -1 & 1 \end{bmatrix}. \qquad [32]$$

Now we can approximate the value of $\varphi(|\nabla u|^2)$ at $x_{i,j}$ by

$$\varphi\left(\frac{1}{2}\left(\left|\frac{u_{i+1,j}-u_{i,j}}{h}\right|^2+\left|\frac{u_{i,j}-u_{i-1,j}}{h}\right|^2\right)+\frac{1}{2}\left(\left|\frac{u_{i,j+1}-u_{i,j}}{h}\right|^2+\left|\frac{u_{i,j}-u_{i,j-1}}{h}\right|^2\right)\right), \qquad [33]$$

where the following relation holds

$$\frac{u_{n+1,j}-u_{n,j}}{h}=\frac{u_{i,n+1}-u_{i,n}}{h}=\frac{u_{1,j}-u_{0,j}}{h}=\frac{u_{i,1}-u_{i,0}}{h}=0. \qquad [34]$$

Let us now define the following difference operators:

$$
\begin{aligned}
D_1^+ &= D^+ \otimes I \\
D_1^- &= D^- \otimes I \\
D_2^+ &= I \otimes D^+ \\
D_2^- &= I \otimes D^-
\end{aligned}
\qquad [35]
$$

which are in order the forward difference in the first variable, the backward difference in the first variable, the forward difference in the second variable, and the backward difference in the second variable. Now we can give a discrete version of the regularization problem given by

$$
J_h(u_h) = |u_h - z_h|^2 + \sum_{i=1}^{n}\sum_{j=1}^{n}\left[\beta_1 \varphi\left(\frac{\left(\left|\left(D_1^+\right)_i u_h\right|^2 + \left|\left(D_1^-\right)_i u_h\right|^2 + \left|\left(D_2^+\right)_j u_h\right|^2 + \left|\left(D_2^-\right)_j u_h\right|^2 \right)}{2} \right) \right. \qquad [36]
$$
$$
\left. + \beta_2 \varphi\left(\left(S_h u_h\right)_{i,j} \right) \right]
$$

The necessary and sufficient optimality condition for J(u) is

$$
u_h + \beta_1 \left(\left(D_1^+\right)^t \Lambda_1 D_1^+ + \left(D_1^-\right)^t \Lambda_1 D_1^- + \left(D_2^+\right)^t \Lambda_1 D_2^+ + \left(D_2^-\right)^t \Lambda_1 D_2^- \right) u_h + \beta_2 S_h \Lambda_2 S_h u_h = z_h, \quad [37]
$$

where $z_h \in R^{n^2}$ is defined by $(z_h)_{i+(j-1)n} = z(x_{i,j})$ and Λ_1 and Λ_2 represent the n^2 by n^2 diagonal matrices with the diagonals defined by

$$
\lambda_{i+(j-1)n} = \varphi'\left(\frac{\left(\left|\left(D_1^+\right)_i u_h\right|^2 + \left|\left(D_1^-\right)_i u_h\right|^2 + \left|\left(D_2^+\right)_j u_h\right|^2 + \left|\left(D_2^-\right)_j u_h\right|^2 \right)}{2} \right), \qquad [38]
$$

and $\mu_{i+(j-1)n} = \varphi'\left(\left(S_h u_h\right)_{i,j}\right)$ respectively.

Matlab Implementation

To implement, we initially use Matlab to solve small-scale problems in their entirety. For a small system, n = 70, we can first create an image (70 by 70 pixels). Then by adding a random variable to this image, create a noise element to produce a noisy initial image. Then using the algorithm developed, we can obtain a solution. In this case, it is feasible to use Matlab's built in commands to solve the linear system since it will do it relatively quickly. Note that a 70 by 70 image results in the need to solve a linear system of 4761 equations. That is in general, an n x m image will result in a linear system of $(n-1)(m-1)$ equations. However a 70 by 70 pixels is a relatively small-scale image, and our goal is to develop an algorithm that can solve a 256 by 256 problem in minimum time.

The following is the code used for the main Matlab calling routine.

```
%function [x,e]=reconst(idx,n,H,mu,g,c,ep,del,bt)

% The first part is an initialization phase (idx==0)

% This first block creates an initial image
if idx==0; m=n+1; l=n-1; n0=l*l; nn=2*n0;
dx=1/50; tmp=[dx:dx:1-dx]'*ones(1,49);
z=min(abs(tmp),abs(1-tmp)); z=min(z,min(abs(tmp'),abs(1-tmp')));
z=min(z,.3);
zz=z; z=zeros(l,l); z(10:58,10:58)=zz; z(:,1:25)=zeros(l,25);

% The next step adds in random noise to the image resulting in a noisy
image
f0=z+2*del*(rand(l,l)-.5*ones(l,l)); f0=f0(:); u=zeros(m,m);

% The next step is to create the 2-D laplacian
d=ones(l,1);
h0=n*n*spdiags([-d 2*d -d],-1:1,l,l);
h0=kron(speye(l),h0)+kron(h0,speye(l)); b1=n*spdiags([-d d],[-1
0],l,l);

% Here we create the forward and backward differences with respect to
each coordinate
b2=kron(b1,speye(l));  % Backward WRT 2nd coordinate
b1=kron(speye(l),b1);  % Backward WRT 1st coordinate
c1=-b1'; % Forward WRT 1st coordinate
c2=-b2'; % Forward WRT 2nd coordinate
```

```
bb=[b1;b2]; cc=[c1;c2];
h=speye(n0)+mu*h0; % This line can be omitted when solving large
systems

% x=h\f0;
x=f0;  % The above line is replace with this line to bypass the Matlab
matrix
       % solve. For larger systems, the matrix solve step is
computationally
       % intense.  This is valid since h is approx. the identity matrix
.
end % End of Initialization Part

% Begin solution phase (idx==1)

% The next step is to calculate the sqrt(abs(grad u)^2)
if idx>0;
q1=b1*x; q2=c1*x; q3=b2*x; q4=c2*x;
q=.5*(q1.^2+q2.^2+q3.^2+q4.^2); k1=find(q<.2); k3=find(q>1);

% This next s
if idx==1; qq=sqrt(ep+q); b=g./qq; end

a=spdiags([b;b],0,nn,nn);
 c=gg/n/n; qq=h0*x; qq=c./sqrt(ep+qq.^2);
aa=spdiags(qq,0,n0,n0);

% The next step is to find the hessian matrix
hes=h0*aa*h0+bb'*a*bb+cc'*a*cc+h; f=f0-hes*x;

% The following is the interface for calling the multigrid solver
n
(n-1)^2
nnz(hes)
dummy=input('Pausing ... edit for new n? 1 yes, 2 no: ');
if dummy==1
   unix('emacs test.f &');
end

dummy=input('Pausing ... recompile for new n');
if dummy==1
unix('f77 test.f amg1r5.o ctime.o');
end

%flag=1
test0007
%flag=2

unix('a.out') % This is the actual solve step...replaces matlab matrix
solve
dx=load('ansu'); x=x+dx;

u(2:n,2:n)=reshape(x,1,1); end % Solution Phase (idx==1)
```

Multigrid

Our problem of speed reduces to the age-old problem of solving a linear system. In addition to getting a robust and accurate answer, speed is now an important factor. We turn to the use an algebraic multigrid method to solving the system. Multigrid is a coarse grid, relaxation method. In multigrid operations, there are typically four steps to the algorithm. The first involves building a coarse grid of the original system and performing relaxation on a fine grid until the error becomes smooth. The relaxation step tends to smooth errors. This dampening effect tends towards more accurate solutions. After computing and then restricting the residual, it is transferred to the coarse grid. Solving the coarse grid residual equation is the next step. Then there is the interpolation of the error and the correction stage.

Algebraic multigrid is a more robust algorithm that does not take advantage of the geometry of a system. Here one can apply multigrid to an unstructured grid, which is impossible to do in the geometric multigrid scheme. In essence, it is the opposite of geometric multigrid. First the algebraic method defines the multigrid components and then it performs the multigrid cycles. More specifically it fixes the relaxation and then defines the multigrid components so that coarse grid correction eliminates error not reduced by relaxations.

There are two phases to algebraic multigrid. The first is the setup phase. It involves setting up the coarse grids and defining interpolation, restriction, and the coarse grid operators. The second step is the solution phase. Here we have the standard multigrid cycling operations.

Fortran AMG

Now that we have described a little about how algebraic multigrid works, we must now put it into practice. First I obtained a recommended algebraic multigrid package from http://www.mgnet.org/mgnet-codes-gmd.html. This package is a Fortran-77 driven multigrid package developed by John Ruge, Klaus Stueben, and Rolf Hempel.

The package we used contained four Fortran files: amg1r5.f, aux1r5.f, ctime.f, drv1r5.f. The first file contained code that was the entire algebraic multigrid process. The second file was a driving interface program that setup parameters, input, and output methods. The last two files are not used since they do not pertain directly to the solution of the system. However ctime.f must be compiled and included when creating the main executable file. drv1r5.f does not need to be included. In using this package, we did not use several available features for ease of use. The main subroutine, amg1r5.f was not changed. The driving program aux1r5.f was changed to fit our needs. Before we get into change, we must first describe how information is stored so that it is usable to this program.

Given a linear system $L \cdot x = f$, we need to store this information in a series of vectors. First there are several assumptions on L. The first is that diagonal elements are always positive on all grids. The second requirement of the program is that L is a square matrix, which is singular with row sums equal to zero. These are requirements of the program. Additionally there are some theoretical restrictions to L. L is a positive definite, most of the off diagonal elements are nonpositive, and row sums are non-negative. Since we are working with large, sparse matrices, we need to develop an efficient storage scheme. The information given by the matrix is stored in a series of

three vectors in what is called "compressed sky-line" fashion. The first, calling it A, contains the non zero elements of L stored by rows, with each row starting with its diagonal element, and the rest of the other non-zero entries following the leading diagonal entry in any order. The last entry of A contains a dummy element, which is needed to match the sizes of the other vectors. It contains no usable information and its value is arbitrary.

Now in order to preserve the locations of the non-zero elements in the sparse array, we create two additional vectors, IA and JA. Now letting NNU represent the number of unknowns in the system and NNA represent the total number of non-zero elements in A, we can define IA. Elements of IA(I) point to the position of the diagonal entry of row I with in the vector A. For example if the third diagonal element is stored in the 6^{th} entry in vector A, then IA(3) contains the value 6. Then theoretically the dimensional of vector A should be NNU. However the program needs the length of vector A to be NNU+1, with the last entry being NNA+1. This last addition allows the program to recognize the length of the last matrix row. To define JA, we first establish that is has the same length as A. That is it contains NNA+1 elements. Remember, when we defined A, there was an extra dummy element attached to the end. Now the Jth element in JA corresponds to the Jth element in A. More specifically, JA(J) points to the column the entry A(J) was in, in the original matrix L. For example if the (4,5) element of matrix L was –3, and that entry was the 9^{th} entry in vector A, then JA(9)=5. Here again, since the length of JA must match that of A, then another arbitrary value must be added for the NNU+1 entry of JA.

Now as it seems, the storage requirements of these vectors are known ahead of time. However, since the routine will actually change these vectors in the multigrid process, we need to allocate much more space than these vectors initially use. The program suggests that the size of the vectors A and JA should be 3*NNA + 5 * NNU while the size of IA, U, and F should be 2.2 * NNU. And the size of IG is 5.4 * NNU. In practice however, we found that IG would often need more allocated memory, so 10 * NNU was used instead. Vector F contains the "right hand side" of the system. Output vectors U, which contains the solution, and IG, which is used internally, are two vectors which space must be allocated to beforehand.

Now we can look at the call to the main subroutine. In the code, the line is

```
 CALL AMG1R5(A,IA,JA,U,F,IG,NDA,NDIA,NDJA,NDU,NDF,NDIG,
+            NNU,MATRIX,ISWTCH,IOUT,IPRINT,LEVELX,
+            IFIRST,NCYC,EPS,MADAPT,NRD,NSOLCO,NRU,
+            ECG1,ECG2,EWT2,NWT,NTR,IERR)
```

The first 6 parameters have been mentioned previously. The next six parameters NDA, NDIA, NDJA, NDU, NDF, NDIG represent the dimensioning of the corresponding vector in the calling program. NNU is again the number of unknowns. MATRIX is a two-digit integer variable that represents what kind of matrix L is. The first digit is a 1 if L is symmetric and 2 if it is not symmetric. The second digit is 1 if L has row sum zero and 2 if it is not. In our use, MATRIX will have a value of 12, a symmetric matrix without rowsum equal to zero.

The rest of the parameters are used to control how multigrid works. EPS is the value for convergence criterion. This value was most critical in finding out the correlation between accuracy and calculation times of the program. ISWITCH is a parameter controlling which modules of AMG1R5 are being used. Here we use a value

of 4, which allows for all four modules to be used. The first module is SETUP, which defines the operators needed in the solution phase. The second is FIRST which initializes the solution vector. SOLVE computes the solution by algebraic multigrid cycling. And the final module, WRKCNT, provides information about residuals, storage requirements, and cycles.

IFIRST is a value that represents the parameter for the first approximation. It is also a two-digit number. The first digit is arbitrary since it is not used and is set to 1, since it must be non-zero. The second digit is 0, 1, 2, or 3. 0 represents no setting of first approximation. 1 represents the first approximations of zero. 1 represents a first approximation of 1. 3 represents the first approximation to be a random function determined by the program.

IOUT is a parameter that represents what output is shown to the user. For our purposes we set this integer value to be 13, which specifies residual output information. IPRINT is a five-digit integer value that represents Fortran unit parameters. We set this to be 10606. IERR is an output parameter, which contains error flags. A value of zero translates to no errors. A negative value will be a non-fatal error, while a positive value represents fatal errors.

LEVELX, NCYC, MADAPT,NRD, ,NRU, ECG1, ECG2, EWT2, NWT, NTR are parameters which have standard values. LEVELX, MADAPT,NRD, ,NRU, NTR have standard values of zero. NCYC has a value of 10250. NWT is 2. ECG1 is 0.D0, ECG2 is 0.25D0, and EWT2=0.35D0. Finally NSOLCO is a parameter that toggles the use of an external package not provided by AMG1R5. We set this parameter equal to 1 to turn this feature off.

Results

In this section, we describe how the proposed method performs using a simulated test example. Many factors can come into testing our proposed algorithm. The first is what are the performance concerns. Obviously there is a degree of accuracy involved. In these types of inverse problems, accuracy is often hard to determine. But since our testing involves an initial image that is polluted with a certain amount of noise, we can have a measure of accuracy. First we create the test image, $z_{i,j}$, as shown in Fig. 4.

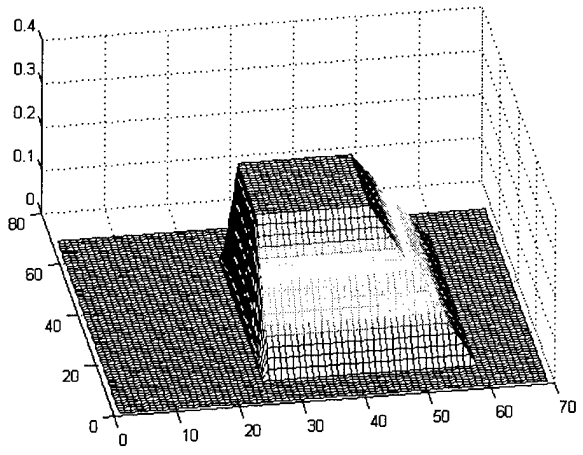

Fig. 4

Then we add the noise to the test image resulting in the observed image, $y_{i,j}$.

$$y_{i,j} = z_{i,j} + \delta n_{i,j}, \qquad [39]$$

where $n_{i,j}$ is an independent, identically distributed random variable with uniform distribution on (-1,1), . This noisy image is shown in Fig. 5.

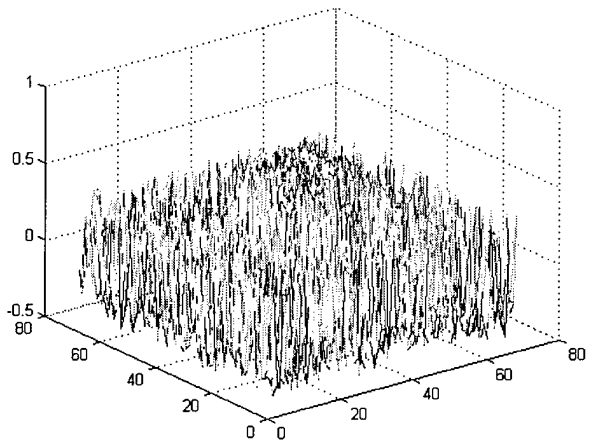

Fig. 5

Accuracy in large part is going to be determined by our choices of constants, β_1 and β_2. These two, as mentioned before in the mathematical discussion, control the edge preserving nature of the method and the shape preserving method. Also a concern is the phenomena of the deterioration of the height (level of gray scale) of the image. Finding an acceptable balance that keeps edges, smoothes slopes, and does not decrease the height of the image is our goal.

In doing so it is necessary to test different values of β_1 and β_2 together. What we observed is that a smaller value of β_1 will result in less of the deterioration phenomena. This effect is shown in Figs. 6, 7, and 8. For each of these examples, $\beta_2 = 0$. The values for β_1 are 0.01, 0.02, and 0.1, respectively. Notice that the larger β_1 is, the blockiness is accentuated. Also as β_1 increases in value, the deterioration effect increases.

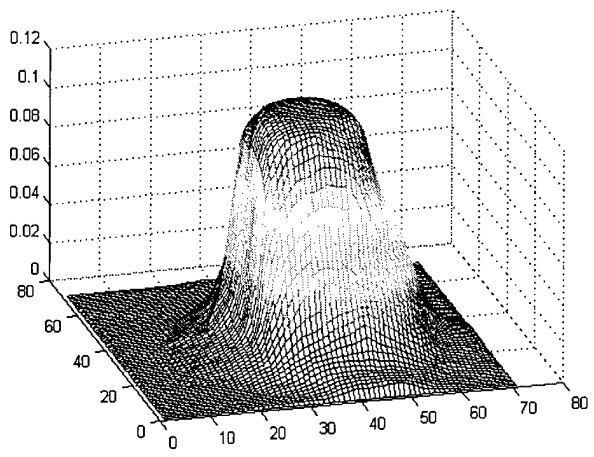

Fig. 6

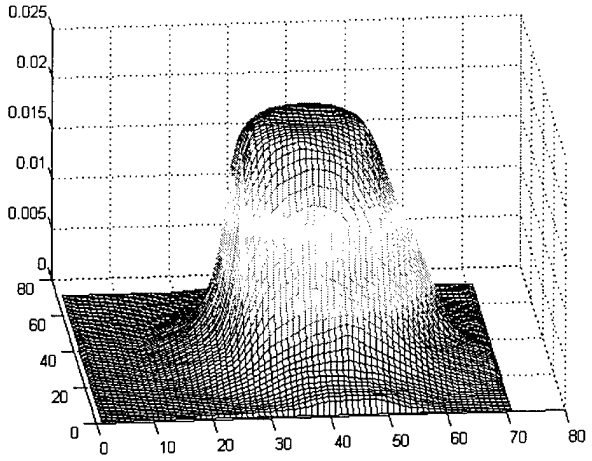

Fig. 7

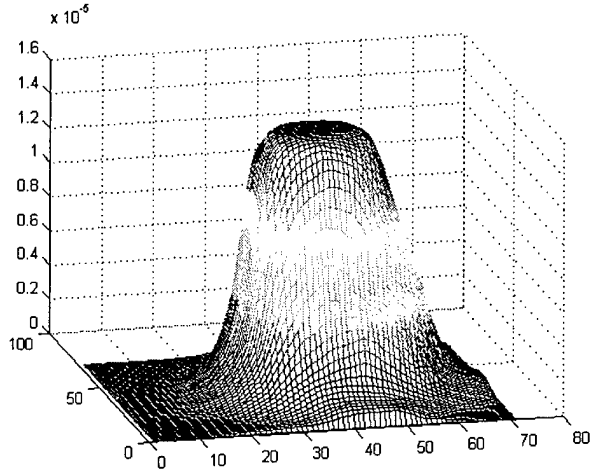

Fig. 8

We now introduce the second BV term. Here we are careful to pick our value of β_2 so that it removes the blockiness in the smooth regions and retards the deterioration effect. Fig. 9 represents the BV regulation method with choice of β_2 =0.007. The values for β_2 are 0.01, 0.05, 0.1 used successively.

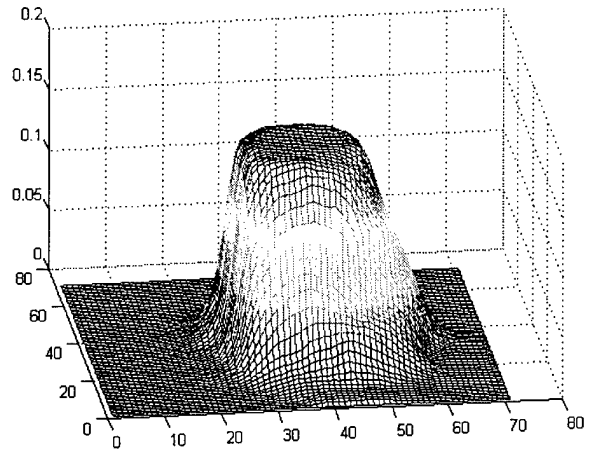

Fig. 9

The parameter values for Fig. 10 are β_1 =0.01. The values for β_2 are 0.01, 0.05, 0.1 used successively.

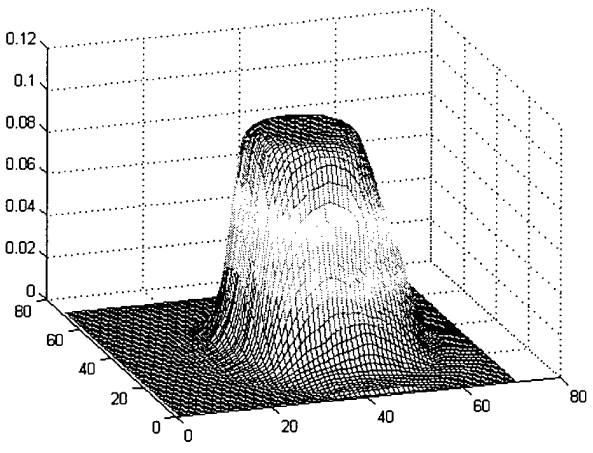

Fig. 10

As one can see, careful choice of parameters will result in a reconstructed image with the three types of image composition. Here we see that increasing β_1 increased edge definition, but the successive choices for β_2 allowed us to retard blockiness and the height deterioration effect.

Another concern is the speed in which an answer is produced. Because the size of the system increases with the square of the number of pixels, large images can result in time-consuming calculation. Our method using multigrid was chosen to decrease calculation times, however careful choice of epsilon may also create significant gains in compute times. An epsilon 10^{-6} vs. 10^{-14} produced the same accuracy. The number of AMG cycles performed was less when we used. This performance is based on a 128 by 128 pixel image. Testing of 256 by 256 was interrupted due to storage limitations. A better interface between the Fortran AMG routines and the Matlab code would remedy this problem.