# Image Segmentation with the Mumford Shah Functional

Russell Valentine

May 6, 2007

**Abstract**

This paper is an introduction to the Mumford Shah (MS) variational based image segmentation method. It describes the main concepts behind the MS, a MS approximation, and shows several numerical solutions as examples.

## 1 Introduction

Images are the marriage of intensity with direction. Images are patterns of light intensity in which the amount of light on any point corresponds to the direction of origin of the light [1].

Summarized from [2], let the function $g(x, y)$ be the function that gives us the intensity of light at point $(x, y)$. Let the function $g(x, y)$ be called an image on our domain $\Omega \in \mathbb{R}^2$. The image is a sample of a scene taken from the three dimensional world. A common problem in computer vision is to find subsets of the domain, which we will call regions that represent different objects in the image. What we want is to find a region $\Omega_1 \in \Omega$ that represents the object $O_1$ in our scene. If a scene has multiple objects $O_1, O_2, O_3, \ldots$ one might want to find the regions of the objects in the scene $\Omega_1, \Omega_2, \Omega_3, \ldots$. If an object $O_2$ is in front of $O_1$ the regions $\Omega_1$ and $\Omega_2$ will share a common boundary (the edge of $O_2$). One would expect $g(x, y)$ to be discontinuous along this shared boundary. A way to model $g(x, y)$, is to get a set of piecewise smooth functions $f_i$ for each object. The idea behind segmentation is to find these piecewise smooth functions $f_i$ and thus the region $\Omega_i$ from the general function $g(x, y)$.

There are several uses for image segmentation. In medical imaging one might use image segmentation to identify tumors, organs, or blood vessels. One can locate objects in satellite images such as roads, forests or buildings. Face recognition, traffic control, or vision in robots can all benefit from image segmentation.

There are several approaches to image segmentation, this paper will cover one of them– image segmentation with the Mumford Shah functional.

# 2 Mumford Shah

The Mumford Shah (MS) was introducted by Mumford and Shah in 1989 [2]. It follows:

$$E(f,\Gamma) = \mu^2 \int\int_{\Omega} (f-g)^2 \, dxdy + \int\int_{\Omega-\Gamma} \|\nabla f\|^2 \, dxdy + \nu L(\Gamma) \quad (1)$$

Like before, $g$ is our image function. We have $\Omega = \Omega_1 \cup \Omega_2 \cup \cdots \cup \Omega_k \cup \Gamma$ in which $\Omega$ is the domain of our image, $\Omega_i$ is the region in our image that represents a object $O_i$ which does not including the boundaries, and $\Gamma$ is the set of smooth arcs that make up boundaries for the $\Omega_i$. $L(\Gamma)$ is the total length of all the smooth arcs in the set $\Gamma$. The function $f$ is a function that is differentiable on $\Omega_i$ $i = 1, 2, \ldots, k$ but can be discontinuous across $\Gamma$. The values of $\mu$ and $\nu$ are weighting factors that control the quality of approximation and coarseness of the segmentation– a large $\nu$ will result in fewer boundaries [3] (See Figure 2 for an example).

The goal is to find $f$ and $\Gamma$ so that $E$ is minimized and it is proposed that a smaller $E$ gives us a better segmentation than a bigger $E$. The first term makes $f$ approximate $g$, the second term insures that our regions $\Omega_i$ does not change drastically (they are regions instead of boundaries or another region), and the third term makes the boundaries $\Gamma$ as short as possible. It should be noted that removing any of these terms gives us a trivial solution $\inf E = 0$. Remove the first term and $f = 0$ making $\Gamma = \emptyset$. Remove the second $f = g$, with $\Gamma = \emptyset$. The removing of the third term means $\Gamma$ can be any set such as a grid of many horizontal and vertical lines with each block being infinitesimally small such that $f(x,y) = g(x,y)$ and $\|\nabla f\|^2 = 0$.

The resulting function $f$ ends up becoming a simplified version of the original $g$, similar to making a cartoon out of a photograph. The cartoon has less texture but its main features still exist. What was perceived as a road in the photograph is still perceived as a road in the cartoon.

The MS is far from perfect. Because it relies on boundaries there are many instances such as rough textures, reflections, or shadows that can cause problems.

The MS was proposed in [2] and contained the first steps toward a solution, however it was mainly a conjecture. It is suggested that those interested in various existence proofs for certain cases of the MS refer to [5]. The MS has shown to be a successfully applied model. It should be noted that since $E$ lacks differentiability one can not use the Euler-Lagrange equations to solve it.

One case that has a complete mathematical analysis is the simplified Mumford and Shah functional [6]:

$$E(f,\Gamma) = \int\int_{\Omega-\Gamma} \|f-g\|^2 \, dxdy + \nu L(\Gamma) \tag{2}$$

We exclude the trivial solutions. This is what will be solved using an algorithm explained in the next section. Note that in eq. (1) if $f$ is a constant piecewise function $\int\int_{\Omega-\Gamma} \|\nabla f\|^2 \, dxdy = 0$.

# 3 Region Growing Algorithm

There are many algorithms for segmenting images using forms of the Mumford and Shah. The algorithm we shall study is a region merging algorithm explained in [6] and [3]. Using the simplified MS eq. (2), we start by getting a list of all possible small regions. We make $f$ piecewise constant. We then take two adjacent regions – $\Omega_i$ and $\Omega_j$ – and see if the functional $E$ is smaller when they are combined. Calculating the difference with (For derivation see [4]):

$$\begin{aligned} \Delta E &= E(f,\Gamma) - E\left(\tilde{f}, \Gamma - \delta(\Omega_i,\Omega_j)\right) \\ &= \frac{|\Omega_i|\cdot|\Omega_j|}{|\Omega_i|+|\Omega_j|} \cdot \|\beta_i - \beta_j\|^2 - \nu \cdot L(\delta(\Omega_i,\Omega_j)) \end{aligned} \tag{3}$$

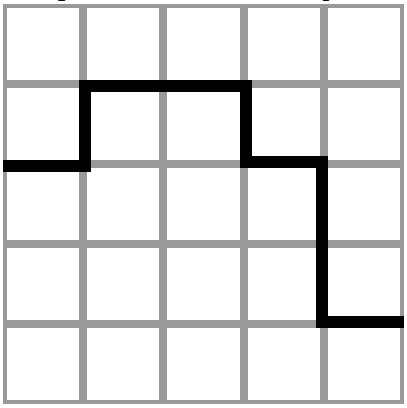
We have $f_i$ that is a constant function equal to the average pixel value in $g$ for that region $\Omega_i$. For example, let $\beta_i$ be the mean intensity value of $g$ in the region $\Omega_i$, then $\forall(x,y) \in \Omega_i; \quad f(x,y) = f_i(x,y) = \beta_i$ . We have $|\cdot|$ being the area measure and so $|\Omega_i|$ is the area of the region $\Omega_i$, $\|\beta_i - \beta_j\|^2$ is the absolute value of the difference in the constant values that make up the functions $f_i$ , $f_j$ that approximate $g$ on their appropriate regions. $L(\delta(\Omega_1,\Omega_2))$ is the length of the boundary that changes when $\Omega_i$ merges with $\Omega_j$, and $\nu$ is the weight constant that was mentioned in the previous section. Once we have found that merging adjacent

regions into $\Omega_i$ no longer lowers our functional $E$, we move to the next region and continue again. We continue until there are no more regions in which merging lowers our functional $E$.

Most often when dealing with image segmentation we are using digital images (discrete images). Up until now we have been treating our domain as a continuous image. Digital images are discrete and have a domain $\Omega = (x, y) \in \mathbb{Z}^2$ where $x \in (1, n)$ and $y \in (1, m)$ and $n$ by $m$ is the resolution of the image. One might think of it as taking a $n$ by $m$ grid of samples of a scene from the continuous world, where we have a measured intensity of light at data points but no data in between each adjacent point. From now on we will be discussing this algorithm specifically for digital images.

In our region merging algorithm on a digital image, we can start out by having each pixel in our image represent a region, and so there are $n \cdot m$ regions to start with. We then merge pixels if they lower our functional $E$ by calculating $\Delta E$ given in eq. (3). The area $|\Omega_i|$ can be calculated by adding up the number of pixels in the region. The value $\beta_i = \frac{\Sigma_{x,y}^{\Omega_i} g(x,y)}{|\Omega_i|}$ can also be easily calculated. The boundary length $L(\delta(\Omega_i, \Omega_j)) = L(\Omega_i) + L(\Omega_j) - L(\Omega_i \cup \Omega_j)$ . Each length $L(\Omega_k)$ can be measured in a discrete manner by going along the outside of the outer pixels of a region and finding the perimeter as shown Figure 1. We now have all we need to implement the algorithm to segment digital images. To see an implementation of this algorithm written for Octave see Appendix A.

Figure 1: Border of length 9 in black, separating two areas one of area 10 and the other of area 15, pixels are represented as white squares.



# 4 Results

The Figures 2, 3, and 4 are results of different images $g$, their approximate piecewise function $f$, and a representation of the borders $\Gamma$ for various values of $\nu$. They were obtain by using the Octave implementation from Appendix A which implements the region growing algorithm of the simplified MS eq. (2) as mentioned in the previous section.

Figure 2: Original image $g$, $f$, and border $\Gamma$ with $\nu = 5$

$\nu = 10$, and $\nu = 15$

$\nu = 20$

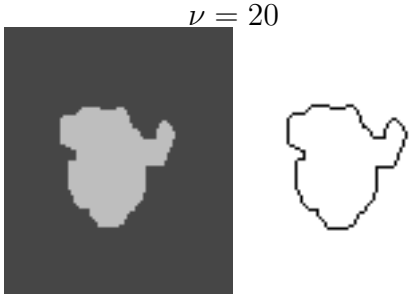


Figure 3: $\nu = 5$ and $\nu = 10$

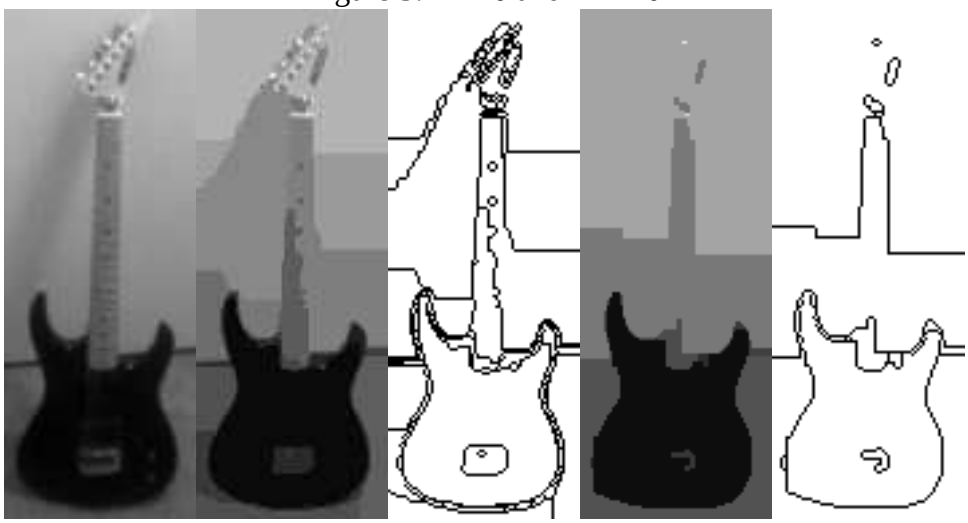


Figure 4: $\nu = 13$

# 5 Conclusion

The segmentation is often not able to differentiate between objects and a shadow as this is a inherent problem in the MS. As you can see in Figure 2 with $\nu = 25$ we almost segment the entire face from the rest of the image and with $\nu = 10$, the eyes, mouth, and eyes have some segmentation, it is conceivable with a higher resolution image and a mug shot the algorithm discussed in this paper can possibly be used in facial recognition as well as other applications.

The implemented algorithm (Appendix A) is quite slow– the images in this paper took many hours each to segment on a modern desktop computer. Rewriting the implementation in a faster language as well as improving the "regionBorderLength()" and "mergeRegions()" functions should improve the speed dramatically. The reference [3] offers a method to improve the speed called "Tiling" which decides the sequence of adjacent regions to work on. Also a study of the implementation in [3] could be done to see if that implementation improves speed.

Another topic to look into is to find a definitive way of choosing $\nu$ to get the segments you want out of the image besides just guess and checking.

This paper only mentions segmenting images with one channel, better segmentation can happen if a multi-channel image is used and if the implementation handles it correctly. Having multiple channels gives you more information. Example of multi-channel images are color images in which each channel represents the image for a different range of wavelength of visible light (such as in a three channel Red, Green, Blue image). It is possible to segment using the information from multiple channels and is discussed in [3] and [6].

There are many other approximations of the MS besides the simplified MS eq. (2) and other algorithms besides region growing. These approximations and algorithms might be better suited depending on our segmentation problem. See [5] for a place to begin on this.

The reference [7] discusses using the MS as a way to get a higher resolution image out of a set of images (such as a video) and using the MS for interpolation. Other uses of the MS besides segmentation could be looked at.

# References


[1] R. Berry and J. Burnell. Astronomical Image processing. Willmann-Bell, Inc. Virginia. 2002. Third Printing.

[2] D. Mumford and J. Shah. Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems. Communications on Pure and Applied Mathematics, Vol. 42, pp 577-686, 1989.

[3] J. Martin. Implementing the Region Growing Method Part 1: The Piecewise Constant Case. Naval Air Warfare Center Weapons Division. China Lake, California. 2002 (NAWCWD TP 8525, publication UNCLASSIFIED).

[4] J. Martin. A General Merging Criterion. Naval Air Warfare Center Weapons Division. China Lake, California. 2002 (NAWCWD TP 8350, publication UNCLASSIFIED).

[5] G. Aubert and P. Kornprobst. Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations. Springer Science+Business Media, LLC. 2006. pp 105-173.

[6] G. Koepfler, C. Lopez and J. Morel. A multiscale Algorithm for Image Segmentation by Variational Method. SIAM Journal of Numerical Analysis, vol 31, pp 282-299, 1994.

[7] Todd Wittman. Mathematical Techniques for Image Interpolation. Department of Mathematics, University of Minnesota. <http://www.math.umn.edu/~wittman/Poral2.pdf> July 2005.

# Appendix A: Algorithm Implimentation

```
% Algorithm Implementation for MATH648 Project
% written by Russell Valentine
% Requirements: Octave with Octave-Forge extentions
% See: http://www.gnu.org/software/octave/
%
% This Algorithm is a region merging algorithm
% using the simplified Mumford Shah.
%
% See: G. Koepfler, C. Lopez and J. Morel. A multiscale Algorithm for
% Image Segmentation by Variational method. SIAM Journal of Numerical
% Analysis, vol 31, pp 282-299, 1994.
%
% Example Usage:
% nu=10;
% filename="myimage";
% more off;
% [image1, regions]=segmentPNG([filename,".png"], nu, 1);
% img=double(image1);
% f=getF(regions, image1);
% b=getBorder(regions)*255;
% saveimage([filename,"-region-",int2str(nu),".img"], regions, "img");
% pngwrite([filename,"-grayG-",int2str(nu),".png"], img,img,img,
%          ones(size(img))*255);
% pngwrite([filename,"-f-",int2str(nu),".png"], f,f,f,
%          ones(size(f))*255);
% pngwrite([filename,"-border-",int2str(nu),".png"], b, b, b,
%          ones(size(b))*255);
%


% usage: img=makeGray(image1)
```

```
%
% Takes a RGB color image and returns a grayscaled version since we
% are only operate on single channels in this implementation.
% Arguments:
%   image1      the RGB color image a n x m x 3 array.
%
function img=makeGray(image1)
    img=(0.3*image1(:,:,1)+0.59*image1(:,:,2)+0.11*image1(:,:,3));
endfunction


% usage: area=regionArea(regionNumber, regions)
%
% Returns the number of pixels a region is covering.
% Arguments:
%   regionNumber    The region number to calculate the area for.
%   regions         The matrix containing the regions
function area=regionArea(regionNumber, regions)
    dim=size(regions);
    area=sum(sum(regions==regionNumber));
endfunction


% usage: sG=sumG(regionNumber, regions, image1)
%
% Returns the sum of pixel values in the actual image for a
% region.
% Arguments:
%   regionNumber    The region to sum over.
%   regions         A matrix containing the regions
%   image1          The actual image matrix g
%
function sG=sumG(regionNumber, regions, image1)
    sG=sum(sum(image1(regions==regionNumber)));
endfunction


% usage: length1=regionBorderLength(regionNumber, regions)
%
% Returns the length of the board of a region.
% Arguments:
%   regionNumber    The region to find the board length
%   regions         A matrix containing the regions
%
function length1=regionBorderLength(regionNumber, regions)
    dim=size(regions);
    length1=0;
    for x=1:dim(1)
        for y=1:dim(2)
            if(regions(x,y)==regionNumber)
                if((x>1) && (regions(x-1,y) != regionNumber))
                    length1++;
                endif
                if((x<dim(1)) && (regions(x+1,y) != regionNumber))
                    length1++;
                endif
                if((y>1) && (regions(x,y-1) != regionNumber))
```

```
                    length1++;
                endif
                if((y<dim(2)) && (regions(x,y+1) != regionNumber))
                    length1++;
                endif
            endif
        endfor
    endfor
endfunction


% usage: regions=defaultRegions(dim)
%
% Returns a default region matrix which each pixel is a region.
% Arguments:
%   dim     The size of our image
%
function regions=defaultRegions(dim)
    regions=zeros(dim);
    count=0;
    for x=1:dim(1)
        for y=1:dim(2)
            regions(x,y)=count;
            count++;
        endfor
    endfor
endfunction


% usage: [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
%                   regions, regionNum1, regionNum2, image1, nu,
%                   region1Stuff)
%
% Returns the calculated \Delta E and updates current working region
% data to save on work for next time we calculate \Delta E.
% Arguments:
%   regions         A matrix with our regions
%   regionNum1      Our current working region
%   regionNum2      The region we are merging with to see if it improves
%                   \Delta E
%   image1          Our image matrix g
%   nu              The weight \nu parameter
%   region1Stuff    A matrix contains the current working region data
%                   it should include: [area1, sumg1, f1, borderLength1]
% Inside region1Stuff:
%   area1           The current area for our working region
%   sumg1           The sum of values from the image for our region
%   f1              sumg1/area
%   borderLength1   The length of the border for our working region
%
function [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
          regions, regionNum1, regionNum2, image1, nu,
          region1Stuff)
    area1=region1Stuff(1);
    sumg1=region1Stuff(2);
    f1=region1Stuff(3);
    borderLength1=region1Stuff(4);
```

```
    area2=regionArea(regionNum2, regions);
    sumg2=sumG(regionNum2, regions, image1);
    f2=sumg2/area2;
    oldlb2=regionBorderLength(regionNum2, regions);
    newRegions=regions;
    newRegions(newRegions==regionNum2)=regionNum1;
    newlb=regionBorderLength(regionNum1, newRegions);
    L=borderLength1+oldlb2-newlb;
    normfs=abs(f1-f2);
    dE=((area1*area2)/(area1+area2))*normfs-nu*L;
    if(dE<0)
        area1=area1+area2;
        sumg1=sumg1+sumg2;
        f1=sumg1/area1;
        borderLength1=newlb;
        regions=newRegions;
    endif
endfunction


% usage: regions=mergeRegions(regionNumber, regions, image1, nu)
%
% Goes through and checks to see if a given region should merge with
% any of its surrounding regions. It returns the resulting region
% matrix from any merges.
% Arguments:
%   regionsNumber   The given region to see if any of its adjacent
%                   regions should be merged with it.
%   regions         A regions matrix
%   image1          Our image matrix g
%   nu              The weight parameter \nu
%
function regions=mergeRegions(regionNumber, regions, image1, nu)
  if(sum(sum(regions==regionNumber)) > 0)
    dim=size(image1);
    %Reset already tried if a region was added
    regionAdded=1;
    loop=0;
    area1=regionArea(regionNumber, regions);
    sumg1=sumG(regionNumber, regions, image1);
    f1=sumg1/area1;
    borderLength1=regionBorderLength(regionNumber, regions);
    while(regionAdded)
      alreadyTried=[];
      regionAdded=0;
      for x=1:dim(1)
        for y=1:dim(2)
          if(regions(x,y)==regionNumber)
            if((x>1) && (regions(x-1,y)!=regionNumber) &&
               (sum(alreadyTried==regions(x-1,y)) == 0))
              regionNum2=regions(x-1,y);
              [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
                regions,regionNumber, regionNum2, image1, nu,
                [area1, sumg1, f1, borderLength1]);
              if(dE<0)
                alreadyTried=[];
                regionAdded=1;
```

```
            else
              alreadyTried=[alreadyTried,regionNum2];
            endif
          endif
          if((x<dim(1)) && (regions(x+1,y)!=regionNumber) &&
             (sum(alreadyTried==regions(x+1,y)) == 0))
            regionNum2=regions(x+1,y);
            [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
              regions,regionNumber, regionNum2, image1, nu,
              [area1, sumg1, f1, borderLength1]);
            if(dE<0)
              alreadyTried=[];
              regionAdded=1;
            else
              alreadyTried=[alreadyTried,regionNum2];
            endif
          endif
          if((y>1) && (regions(x,y-1)!=regionNumber) &&
             (sum(alreadyTried==regions(x,y-1)) == 0))
            regionNum2=regions(x,y-1);
            [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
              regions,regionNumber, regionNum2, image1, nu,
              [area1, sumg1, f1, borderLength1]);
            if(dE<0)
              alreadyTried=[];
              regionAdded=1;
            else
              alreadyTried=[alreadyTried,regionNum2];
            endif
          endif
          if((y<dim(2)) && (regions(x,y+1)!=regionNumber) &&
             (sum(alreadyTried==regions(x,y+1)) == 0))
            regionNum2=regions(x,y+1);
            [dE, area1, sumg1, f1, borderLength1, regions]=deltaE(
              regions,regionNumber, regionNum2, image1, nu,
              [area1, sumg1, f1, borderLength1]);
            if(dE<0)
              alreadyTried=[];
              regionAdded=1;
            else
              alreadyTried=[alreadyTried,regionNum2];
            endif
          endif
        endif
      endfor
    endfor
  endwhile
 endif

endfunction


% usage: regions=consolidateRegions(regions)
%
% Returns a matrix of regions in which the region numbers are
% consolidated. For example after merging has been done the
% regions will be numbered 3, 200, 450, etc. This function renames
% them 0, 1, 2, ... We do this so it is easier to use the regions
```

```
% matrix in the future as there will be no gap in the region numbers.
% Arguments:
%   regions     A regions matrix
%
function regions=consolidateRegions(regions)
    ur=sort(unique(regions)).';
    count=0;
    for i=ur
        if(i != count)
            regions(regions==i)=count;
        endif
        count++;
    endfor
endfunction


% usage: f=getF(regions, image1)
%
% Returns our f function which is a approximation for our image g.
% It is a matrix the same size as our image.
% Arguments:
%   regions     A regions matrix
%   image1      The image g
function f=getF(regions, image1)
    f=zeros(size(image1));
    for r=sort(unique(regions)).'
        f(regions==r)=sumG(r, regions, image1)/regionArea(r, regions);
    endfor
endfunction


% usage: boarder=getBorder(regions)
%
% Returns a border matrix the same size as our image g. It is a
% matrix where 1 is where the border is and 0 is not the border.
% The border is our \Gamma.
% Arguments:
%   regions     A regions matrix
%
function border=getBorder(regions)
    border=ones(size(regions));
    dim=size(regions);
    for regionNumber=sort(unique(regions)).'
        for x=1:dim(1);
            for y=1:dim(2);
                if(regions(x,y)==regionNumber)
                    if((x>1) && (regions(x-1,y) > regionNumber))
                        border(x,y)=0;
                    endif
                    if((x<dim(1)) && (regions(x+1,y) > regionNumber))
                        border(x,y)=0;
                    endif
                    if((y>1) && (regions(x,y-1) > regionNumber))
                        border(x,y)=0;
                    endif
                    if((y<dim(2)) && (regions(x,y+1) > regionNumber))
                        border(x,y)=0;
```

```
                    endif
                endif
            endfor
        endfor
    endfor
endfunction


%usage: [image1, regions]=segmentPNG(file, nu, statusOutput)
%
% Returns the grayscale intensity n by m image g and a segmented
% region matrix. This should be the function that gets called first.
% f and \Gamma (The border) could be generated with just these two,
% the image and the regions).
% Arguments:
%   file            The filename of a png file to segment
%   nu              The weight parameter \nu
%   statusOutput    0 to not display anything while working, 1 to
%                   display percentage complete after going through a
%                   region. The percentage is not accurate as the
%                   later regions should move much faster. Also some
%                   regions would have been absorbed while working on
%                   a previous region, but it does give you a little
%                   idea of how far along you are.
%
function [image1, regions]=segmentPNG(file, nu, statusOutput)
    image1=pngread(file);
    image1=makeGray(image1);
    regions=defaultRegions(size(image1));
    numRegions=max(unique(regions));
    for r=0:numRegions
        regions=mergeRegions(r, regions, image1, nu);
        if(statusOutput > 0)
            printf("%d%% Complete\n", (r/numRegions)*100)
        endif
    endfor
    if(statusOutput > 0)
        printf("Consolidating Regions...\n")
    endif
    regions=consolidateRegions(regions);
endfunction
```