Texture Based Image Segmentation using Curve Evolution

by

Jason Chang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2009

Author .....
Jason Chang
Department of Electrical Engineering and Computer Science
January 27, 2009

Certified by .....
John W. Fisher III
Principal Research Scientist
Thesis Supervisor

Texture Based Image Segmentation using Curve Evolution

by

Jason Chang

Submitted to the Department of Electrical Engineering and Computer Science
on January 27, 2009, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

This thesis project will focus on investigating extensions and improvements to the curve evolution and non-parametric density estimate based image segmentation approach proposed by Kim et al. [13]. One main problem with the algorithm proposed by Kim was that it was based solely on the individual scalar pixel intensities and did not take advantage of the underlying structure of the image. In this project, we will try to extend this scalar based segmentation approach to a vector based method. By representing each pixel with a vector quantity, we can capture more of the image's structure rather than just the pixel intensity. We will be applying a texture measure, such as the Steerable Pyramid [9, 24, 23], to generate this vector quantity at every pixel of the image. This extension will cause a few more issues to surface, including those of computation time and sparsity of data. After addressing these issues, we will attempt to segment the vector image in a similar fashion as [13], by maximizing the mutual information between the vector at a pixel and the current labeling.

Thesis Supervisor: John W. Fisher III
Title: Principal Research Scientist

Contents

List of Figures

Chapter 1

Introduction

In the field of computer vision, image segmentation describes the process of distinguishing objects within an image from each other and the background. The problem of image segmentation is important in many applications. For the end-user, if a very robust image segmentation algorithm existed, it could potentially help in many fields.

In medicine, radiologists are doctors trained and dedicated to analyze medical images such as x-rays, MRIs, and CT scans. They work to identify tumors or problematic areas within the medical images to diagnose a patient. Sometimes, the decision of whether or not a tumor exists can only be decided by a trained expert. Image segmentation algorithms could help in medicine by finding irregularities in images when doctors are busy or not present. Even if they were not completely reliable, the algorithms could at least provide a starting point to identifying a problem.

In addition to medicine, image segmentation is important for computer vision as a whole. Many vision applications deal with the recognition and processing of images within an object. The first step to systems such as these is to actually find the object of interest, which is exactly the what image segmentation addresses. Once the objects of an image are identified, processing can be done much easier. The motion of these identified objects within a video could even be used to compress videos with a better algorithm than what currently exists.

The applications and usefulness of image segmentation are very vast. However, the problem of image segmentation is ill-posed [2, 15], meaning that there are multiple

solutions in an unconstrained environment. Algorithms can overcome this obstacle by introducing a set of constraints so that a unique optimal solution exists. This solution would only be optimal while the constraints are met, and thus a good set of constraints is extremely important when developing a segmentation algorithm. There has been much work done on image segmentation in the past. Some of these algorithms (such as [19, 7]) rely on training data. However, when the algorithm is tested and trained with a specific set of images, it may not necessarily do well for images that do not fit into the categories of the training set. Many other algorithms (such as [3, 7, 17]) focus on identifying edges to mark the boundaries of regions. In images that contain a textured object with many edges (e.g. wood grain, zebras, jaguars, etc.), an edge detection based approach usually identifies parts of the texture as separate regions, when they are really from the same object. There are also a set of algorithms (such as [5, 6, 27]) that use a similar approach to [13], but with a parametric model or estimate. However, using a non-parametric estimate allows for a wider range of distributions to be represented. If a parametric model is used, and the observed data is not actually drawn from the model, the estimate may be very inaccurate.

This thesis will focus on improving and extending the well formulated algorithm posed by Kim, Fisher, Yezzi, Cetin, and Willsky in [13] by incorporating a texture measure into the segmentation criterion. This algorithm was chosen as a basis for its robustness and previous success in segmenting images. It also has a very natural extension to a vectorized segmentation, which will be used with a texture measure to improve segmentation results. Thus, the proposed algorithm will combine the already successful results of [13] with a texture measure to improve on the kinds of images it can successful segment. The preliminary results from this new algorithm show significant promise. After the improvements and extensions are finished, we will compare the results empirically with other leading algorithms.

Chapter 2

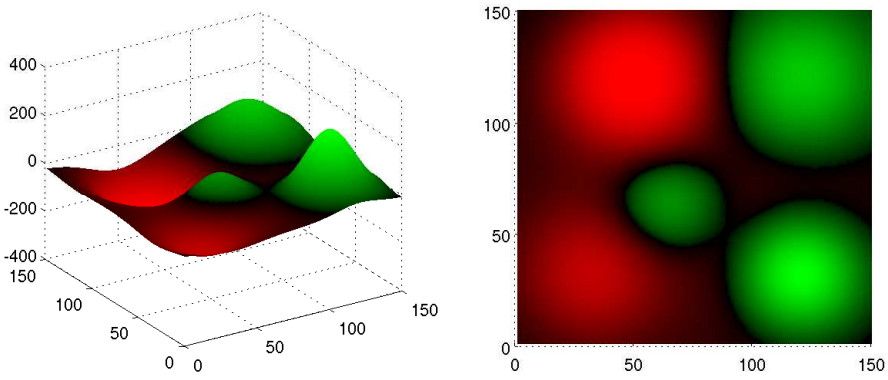
Background Material

Three main topics are necessary in understanding the basis of the algorithm described in this thesis: level set methods, non-parametric kernel density estimates, and texture measures such as the Steerable Pyramid. They will be briefly discussed here, and combined in Chapters 3 and 4 to construct the new algorithm.

2.1 Curve Evolution - Level Set Methods

Level set methods provide a way to implicitly represent and evolve an N-dimensional (or less) hyper-surface in an N-dimensional space. The works of Osher and Fedkiw [18] and Sethian [22] provide the original development of level set methods and provide a wealth of knowledge on this subject. When applied to image segmentation, a three-dimensional surface, ϕ, is defined by values on a two-dimensional Cartesian grid. In practice, this surface is stored as an image, and the height of the level set function is defined for each pixel in the image. For this reason, it will be convenient to refer to the height of the level set at a specific point as the value of the level set at a particular pixel, or $\phi(x, y)$.

The implicit hyper-surface in this case is just a curve that exists in the two-dimensional support of the image. This implied curve will be called the zero level set, $\mathcal{C}$, which is defined as the set of all pixels on the three-dimensional level set function



(a) Plotted in 3 Dimensions (b) Viewed as an Image

Figure 2-1: Level Set Function Viewed as Terrain

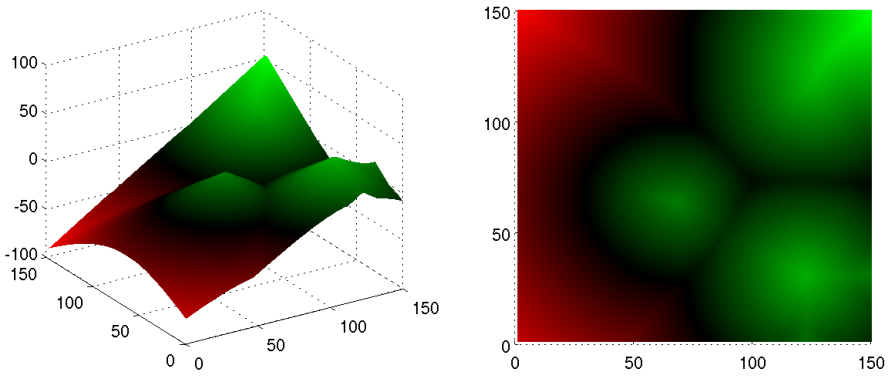
that intersect the plane of value zero.

$$\mathcal{C} = \{(x, y) \mid \phi(x, y) = 0\} \tag{2.1}$$

Thus, the zero level set divides the image into two regions, R^+ and R^-, which consist of the positive and negative values of the level set function respectively. One can often think of a level set function as some sort of terrain, where the land above sea level would be in R^+, the ocean floors below sea level would be in R^-, and sea level would be the zero level set. An example of a level set function is given in Figure 2-1, where the green pixels would be in R^+, the red pixels would be in R^-, and the purely black pixels make up the zero level set. The zero level set will sometimes be referred to as simply "the curve" because it solely determines the final segmentation.

2.1.1 Signed Distance Function

In a typical application of image segmentation with level set methods, the user is only concerned with the zero level set because it is this curve that segments the image. To start the process of evolving a level set function, an initialization of the zero level set, or an initial guess at the segmentation, is needed. The method for choosing this initialization will be discussed in Chapter 3. Assuming the initialization exists, the level set function is then iteratively evolved to refine the guess to a better solution.



(a) Plotted in 3 Dimensions (b) Viewed as an Image

Figure 2-2: Level Set Function as Signed Distance Function

Because the user is only concerned with the zero level set, the values of the level set function off of the zero level set curve are not specifically defined by the user.

A very common approach is to make the level set function a signed distance function. A signed distance function has the property that the absolute value at each pixel is the minimum distance to the zero level set, $\mathcal{C}$. The sign at each pixel depends on which region that pixel belongs to. If the pixel belongs to R^-, the sign is negative, and if the pixel belongs to R^+, the sign is positive. In general, the interior of the zero level set curve is defined to have negative values and the exterioer is defined to have positive values. The signed distance function looks more cone-shaped, and thus, an actual level set function looks more like Figure 2-2 rather than Figure 2-1.

Though a signed distance function does not need to be used for the level set function, it does provide some nice properties. In addition to some more subtle benefits, the main reason why many people use signed distance functions is because:

$$|\nabla \phi| = 1 \tag{2.2}$$

This property is very useful, especially when the velocity depends on a curvature term. This will be explored in more detail in the next section. There are many methods for computing the signed distance function efficiently, but they will not be discussed here. In the implementation, the Fast Marching Method [26] was used.

2.1.2 Evolving the Level Set Function

The typical process for curve evolution is to do the following:

1. Initialize the level set function, ϕ, to a signed distance function with an initial guess
2. Calculate a velocity ϕ_t at every pixel in the level set
3. Update the level set for a small time interval according to ϕ_t
4. Reinitialize the level set function to a signed distance function
5. Repeat from Step 2 until convergence

In level set methods, an energy functional, E, is chosen for the particular application. The problem of calculating the velocity field is then equivalent to minimizing this energy functional. The specific minimization criterion depends on what the goal of the application is. Oftentimes, the energy functional will consist of multiple terms where one of the terms impose some sort of smoothness constraint on the zero level set, making the ill-posed problem of image segmentation more well-posed. The reason for this smoothness constraint will be described more in Chapter 3.

In the algorithm for this thesis, the energy functional will consist of an information theoretic (IT) term and a smoothing (S) term. One way to think of the smoothing constraint is to have a term in the energy functional penalizing curve length. Intuitively, a very jagged curve will have a longer curve length then a smooth curve. Therefore, the energy functional can be rewritten as

$$E(c) = E_{\mathrm{IT}}(c) + E_{\mathrm{S}}(c) = E_{\mathrm{IT}}(c) + \alpha \oint_{\mathcal{C}} ds \qquad \forall c \in \mathcal{C} \tag{2.3}$$

Now, given a specific energy functional, one can minimize this term to find the velocity field. It is important to note that the energy functional is most often only defined on the zero level set, $\mathcal{C}$. This will be discussed in greater detail in Section 2.1.3, but the consequence is that the energy functional only gives a valid velocity for pixels on the zero level set. Ignoring this fact for now, we can rewrite the energy functional due to the curve length penalty in terms of velocities. From the proof in

[11], this velocity is just

$$\overrightarrow{V_{\mathrm{S}}}(c) = \alpha\kappa\overrightarrow{N} \qquad \forall c \in \mathcal{C} \tag{2.4}$$

where κ is the mean curvature given by

$$\kappa = \frac{\triangle\phi}{|\nabla\phi|} \tag{2.5}$$

As previously stated, if ϕ is a signed distance function, then the denominator of κ simplifies to 1, and Equations 2.4 and 2.5 becomes

$$\overrightarrow{V_{\mathrm{S}}}(c) = \alpha\triangle\phi\overrightarrow{N} \qquad \forall c \in \mathcal{C} \tag{2.6}$$

The velocities derived from minimizing the energy functionals are clearly vectors. In the derivation of level set methods in [18], this velocity vector field must be applied to the level set equation.

$$\phi_t + \overrightarrow{V} \cdot \nabla\phi = 0 \tag{2.7}$$

Following the steps in [18, p.42], the equation for the velocity update of the level set just becomes

$$\phi_t(c) = -V_n|\nabla\phi| \qquad \forall c \in \mathcal{C} \tag{2.8}$$

where $\overrightarrow{V} = V_n\overrightarrow{N} + V_t\overrightarrow{T}$. It's interesting to note that velocities from the energy functional are almost always only in the normal direction. The tangential velocities can be safely ignored because they do not change the implicit definition of the zero level set curve. Thus, by combining Equations 2.6 and 2.8, the velocity due to the curve length penalty is

$$\phi_{t,\mathrm{S}}(c) = \alpha\triangle\phi \qquad \forall c \in \mathcal{C} \tag{2.9}$$

Given an energy functional, the updating velocity field due to the curve length penalty can now be calculated. The only thing left in the derivation of evolving the level set is to find the velocity due to the other terms in the energy functional, E_1. This will be done in a Chapter 3 when the entire energy functional is introduced.

2.1.3 Velocity Extension

It was noted in Equation 2.3 that the energy functional (and thus the velocity field) is only well defined for pixels on $\mathcal{C}$, the zero level set curve. However, to actually evolve the level set function, a velocity must be defined for each pixel in the image. One method to define these velocities off the curve is called velocity extension. There are many ways to do velocity extension. One of the most common ways was developed by Adalsteinsson and Sethian [1]. This method constructs the extension velocities in such a way that the signed distance property of the level set function is preserved perfectly after each iteration. However, this method also requires that the velocity be defined for the value at the exact zero level set with sub-pixel accuracy (interpolated through pixels bordering the zero level set). This velocity is not always well defined for images if the pixels used to interpolate belong to separate objects, and thus this method for velocity extension can not be used without further consideration. Instead, a simpler approach to velocity extension (like the one used in [14]) will be used here. In this method for velocity extension, we will just set the value of pixels off of the curve to be equal to the velocity of the closest pixel on the curve. Given the velocities of pixels on the curve, $\phi_t((x,y) \in \mathcal{C})$, we can calculate all the velocities by the following

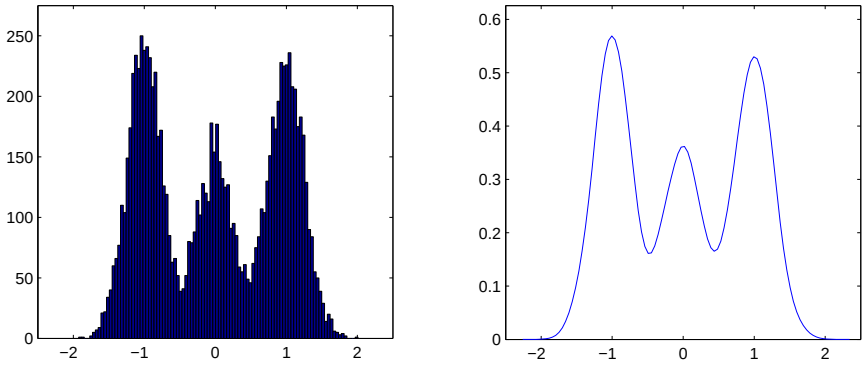
$$\phi_t(x,y) = \begin{cases} \phi_t(x,y) & \text{if } (x,y) \in \mathcal{C} \\ \phi_t(x_{min}(x,y), y_{min}(x,y)) & \text{if } (x,y) \notin \mathcal{C} \end{cases} \tag{2.10}$$

where

$$\begin{aligned} (x_{min}(x,y), y_{min}(x,y)) &= \arg\min_{(x',y') \in \mathcal{C}} \left\{ \sqrt{(x-x')^2 + (y-y')^2} \right\} && (2.11) \\ &= \arg\min_{(x',y') \in \mathcal{C}} \left\{ (x-x')^2 + (y-y')^2 \right\} && (2.12) \end{aligned}$$

2.2 Kernel Density Estimation

In our algorithm, the velocity that evolves the level set depends on the probability density functions of the current segmentation. However, since this PDF is unknown,



(a) Histogram of Samples (b) KDE of the PDF

Figure 2-3: Level Set Function as Terrain

it needs to be estimated. A very common method in estimating density functions is using a Parzen density estimate [20], or what is also known as a kernel density estimate (KDE). The basic concept of the KDE is to smooth the histogram of the current samples with a specific kernel, $K(\mathbf{x})$, to estimate the value of the PDF at any value. An example result of a KDE is shown in Figure 2-3.

For generality, the equations listed here will be for a d-dimensional PDF. The equation for the estimated PDF using a KDE is

$$\hat{p}_{\mathbf{X}}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{h_1 \dots h_d}K\left(\frac{x_1 - x_{i1}}{h_1}, \dots, \frac{x_d - x_{id}}{h_d}\right) \tag{2.13}$$

where d is the dimensionality, N is the number of source points, $\mathbf{x} = [x_1, \dots, x_d]^T$ is the d-dimensional point the PDF is estimated at, $[x_{i1}, \dots, x_{id}]^T$ is the d-dimensional i^{th} source point, and h is the bandwidth of the kernel used to estimate the PDF.

The most commonly used kernel is a Gaussian function:

$$K^G(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}}e^{-\frac{1}{2}\|\mathbf{x}\|^2} \tag{2.14}$$

This KDE gives a good approximation to the actual PDF. However, for large data sets, this computation can take a very long time. Ignoring the dependence on the dimensionality, if the density is needed at M points, then the summation in Equation

2.13 needs to be calculated M times. For each summation, N points are added together. Therefore, the total computational complexity of the KDE is $\mathcal{O}(MN)$. Clearly, this computation could be a bottleneck in an algorithm, especially if the estimation needs to be done multiple times. Thus, a faster method to estimating the PDF must be used. Though the details of the fast algorithms will not be thoroughly addressed, the main concepts of the speedups will be mentioned in the next sections.

2.2.1 The Fast Gauss Transform

In 1991, Greengard and Strain proposed the algorithm called the Fast Gauss Transform (FGT) [12], which approximately evaluates a sum of Gaussians at multiple points in a fast manner.

$$G(\mathbf{x}_j) = \sum_{i=1}^{N} q_i e^{\frac{\left\|\mathbf{x}_j - \mathbf{x}_i\right\|^2}{h}} \tag{2.15}$$

where $\mathbf{x}_j$ is the j^{th} d-dimensional vector that we are calculating the sum at, $\mathbf{x}_i$ is the i^{th} d-dimensional vector source point, and h is the scalar quantity that describes the bandwidth of the kernel. As a note, the bandwidth used in the FGT is constant for all dimensions of the estimation. This may not necessarily be desired. Instead, something like the following may provide for a better estimate

$$G'(\mathbf{x}_j) = \sum_{i=1}^{N} q_i e^{\left\|\frac{\mathbf{x}_j - \mathbf{x}_i}{\mathbf{h}}\right\|^2} \tag{2.16}$$

where $\mathbf{h}$ is now a vector, and the notation of the division in the exponent is abused to mean element-wise vector division. Equation 2.16 can be written more specifically as

$$G'(\mathbf{x}_j) = \sum_{i=1}^{N} q_i e^{\left\|\mathbf{y_{ij}}\right\|^2}, \ \mathbf{y_{ij}} = \frac{\mathbf{x_j} - \mathbf{x_i}}{\mathbf{h}} = \left[\frac{x_j(1) - x_i(1)}{h(1)}, \ldots, \frac{x_j(d) - x_i(d)}{h(d)}\right]^{\mathrm{T}}$$

where $\mathbf{x}_j(a)$ is the a^{th} element in the vector $\mathbf{x}_j$. The details of why this quantity may produce a better estimate, and how to actually compute it will be discussed Section 4.3.2.

To achieve a performance gain, the FGT partitions the sample space into non-overlapping boxes, forming a grid. The algorithm uses these boxes as a clustering of the source and target points. It then uses Hermite and truncated Taylor series expansions to quickly approximate the affect of sources onto the targets. Using this method, the FGT (again ignoring the dimensionality) achieves a theoretical computational complexity of $\mathcal{O}(M+N)$, with a constant factor dependent on the accuracy needed and the bandwidth, h.

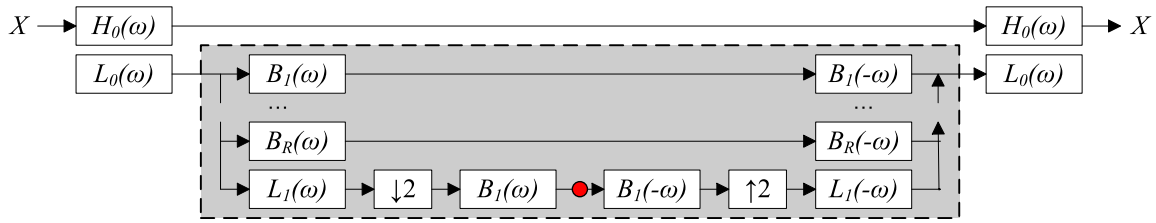
The FGT greatly improves computation times for calculating sums of Gaussians at multiple target points compared to the direct calculation. In low dimensionality problems, the speed of the FGT is very fast. However, the computational complexity increases exponentially with d, the dimensionality. Thus, the FGT is not very efficient in high dimensions.

2.2.2 The Improved Fast Gauss Transform

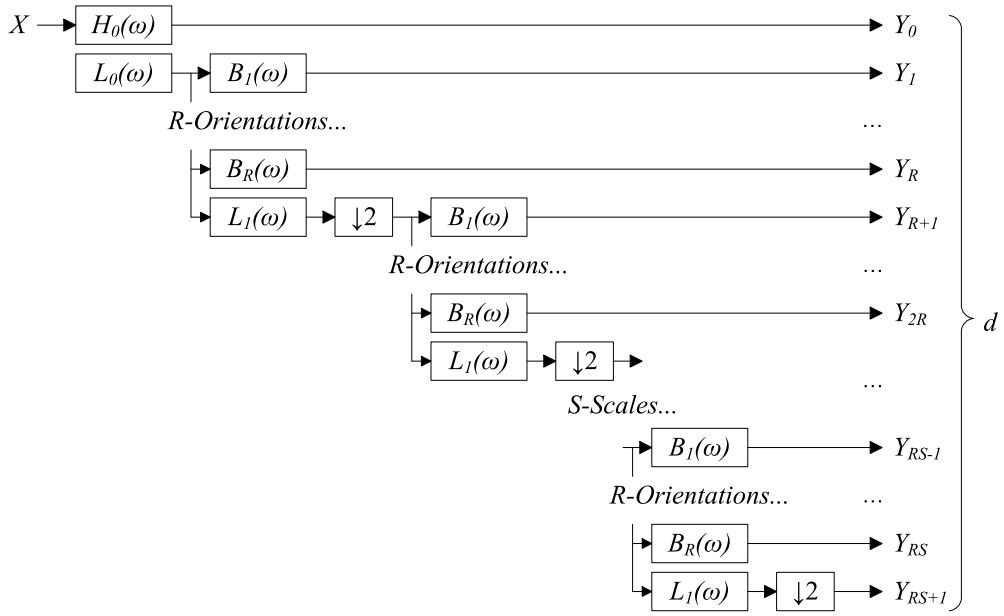
The drawbacks of the Fast Gauss Transform inspired the development of the Improved Fast Gauss Transform (IFGT) in 2003 by Yang, Duraiswami, Gumerov, and Davis [28]. The improvements on the FGT focus on two major points: a better clustering algorithm and the Multivariate Taylor Expansion. The IFGT uses the farthest-point clustering algorithm proposed by Gonzalez [10] to more efficiently divide the source and target points. The Multivariate Taylor Expansion speeds up the Hermite Expansion by decreasing the number of terms in the expansion from a^d to d^p (where a and p are constants). This algorithm is clearly faster than the FGT, and will therefore be used in the segmentation problem.

2.3 Steerable Pyramids

Neighboring pixels within an image are correlated because of the underlying object the pixels are sampled from. Therefore, it is very intuitive that a segmentation based only on pixel intensities without regard for the underlying structure is an oversimplification. Steerable Pyramids provide a multi-scale, multi-orientation decomposition



(a) Steerable Pyramid Recursive Structure: Replace the red dot with an instance of the gray box, and recursively do this until no more down sampling can be done



(b) Steerable Pyramid Outputs at Orientation and Scales

Figure 2-4: Steerable Pyramids Structure and Outputs

of an image. Some good references for Steerable Pyramids are [9, 24, 23].

A Steerable Pyramid uses a polar-separable filter to decompose the image into a number of orientation bands, R, at a number of scales, S. The basic idea of a Steerable Pyramid is that the output of the polar-separable filter oriented at any angle or scale can be determined by interpolating images in the pyramid. Thus, the pyramid provides a complete representation of the image because any oriented or scaled output of the polar-separable filter can be found. This implies that it would be a good way to represent textures within the image.

In addition, the Steerable Pyramid provides both a decomposition and a reconstruction of the image. Figure 2-4(a) provides a block diagram of the pyramid. Notice that both the analysis and synthesis stage of the pyramid can be done by a linear

transformation. The filters in the reconstruction stage are simply the frequency reversed versions of the analysis stage because of the self-invertibility property of the Steerable Pyramid. Thus, the formulation of the pyramid can be written as follows:

$$\mathbf{y}_i = \Phi_i \mathbf{x} \tag{2.17}$$

where the vectors, $\mathbf{x}$ and $\mathbf{y}_i$ (from Figure 2-4(b)), are the 2-D image matrices, X and Y_i, arranged in a column, and Φ_i is a linear transform corresponding to the filters in the pyramid that are used to generate Y_i (i.e. Φ_0 is the linear transform corresponding to the filter $H_0(\omega)$). Thus, the synthesis stage is just:

$$\mathbf{x} = \sum_{i=0}^{d-1} \Theta_i \mathbf{y}_i \tag{2.18}$$

where Θ_i performs the inverse operation of Φ_i (i.e. Θ_0 is the linear transform corresponding to the filter $H_0(-\omega)$). This relationship will be used later to try and reduce the dimensionality of the Steerable Pyramid.

Chapter 3

Previous Work

The algorithm this thesis will cover is an extension on the algorithm proposed in [13]. This section will go into a few details about the method of the algorithm and make comments about its successes and failures.

3.1 The Algorithm

The original algorithm uses a level set method approach to segment a grayscale image. It draws on the fact that the mutual information between a pixel and the segmentation result is high if the segmentation is correct, and low if it is incorrect. This was the reason why the mutual information was chosen as a segmentation criterion in the energy functional. However, because the actual probability densities are unknown, a KDE is used to estimate the densities. A curve length penalty was also used in the energy functional to prevent jagged curves and many small regions. Therefore, the total energy functional is given by Equation 9 in [13] (and replicated here)

$$E(c) = -|\Omega|\hat{I}(G(c); L_{\mathcal{C}}(c)) + \alpha \oint_{\mathcal{C}} ds \qquad \forall c \in \mathcal{C} \tag{3.1}$$

where $|\Omega|$ is the number of pixels in the image, $G(c)$ is the intensity of the image at pixel c, $\hat{I}$ is the approximate mutual information calculated with PDFs from the KDE, and $L_{\mathcal{C}}(c)$ is the labeling that indicates which region the pixel, c, belongs to.

This energy functional results in the following velocity vector field

$$\overrightarrow{V}(c) = \Bigg[log\frac{\hat{p}_+(G(c))}{\hat{p}_-(G(c))} + \frac{1}{|R_+|}\int_{R_+} \frac{K(G(x)-G(c))}{\hat{p}_+(G(c))}\,dx - \frac{1}{|R_-|}\int_{R_-} \frac{K(G(x)-G(c))}{\hat{p}_-(G(c))}\,dx - \alpha\kappa \Bigg] \overrightarrow{N} \qquad \forall c \in \mathcal{C} \tag{3.2}$$

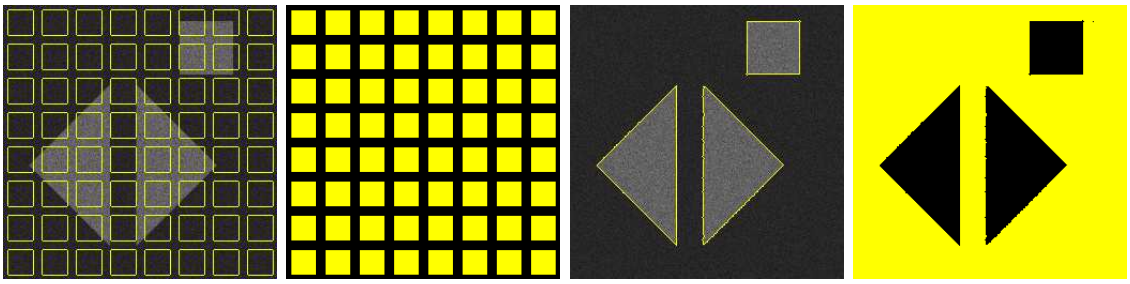
As stated in [13], the second and third term in Equation 3.2 are computational bottlenecks. In most cases, these terms do not affect the segmentation much and can generally be ignored. Therefore, the velocity vector field that was actually used was

$$\overrightarrow{V}(c) = \left[log\frac{\hat{p}_+(G(c))}{\hat{p}_-(G(c))} - \alpha\kappa \right] \overrightarrow{N} \qquad \forall c \in \mathcal{C} \tag{3.3}$$

The paper also presented the formulation of an M-ary segmentation, where M is a power of 2. It uses a multi-phase segmentation like the one presented in [4]. The algorithm shows very good results in the examples images proposed in the paper. In addition, the time needed to segment the image was comparable to other techniques.

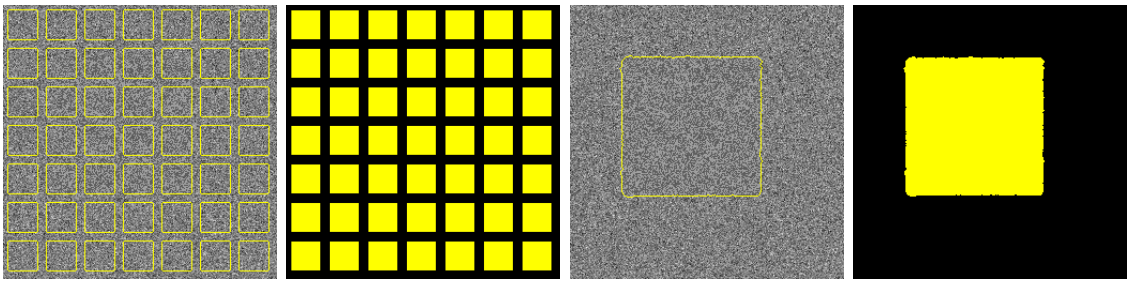
3.2 Results

A few sample images were tested with my own implementation of the algorithm. It was tested with both artificially generated images and images from the Berkeley Dataset [16]. Figures 3-1, 3-2, and 3-3 show three example segmentations on artificially generated images. In each segmentation, (a) shows the initialized zero level set curve, (b) shows the initialized labeled regions, (c) shows the final zero level set curve, and (d) shows the final labeled regions. Note that although Figure 3-2 does not look like it contains two regions, it actually contains a square of bimodal Gaussian PDF and a background of unimodal Gaussian PDF with the same mean and variance. The segmentation is very close to the ground truth, resulting in a very good segmentation. Figures 3-4, 3-5, 3-6, 3-6, and 3-8 show three example images from [16] segmented using the algorithm. The zebra image was done with both a binary and 4-ary segmentation.



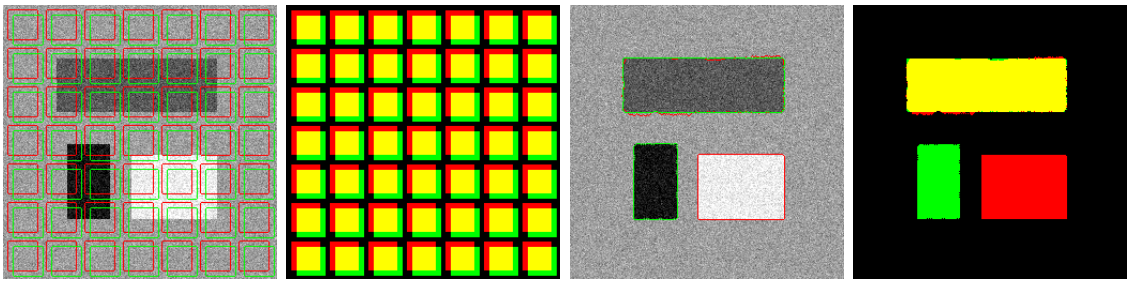
(a) (b) (c) (d)

Figure 3-1: Artificial Binary Segmentation (Different Mean and Variance)



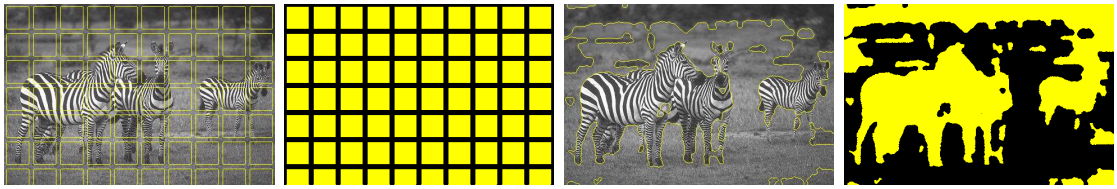
(a) (b) (c) (d)

Figure 3-2: Artificial Binary Segmentation (Same Mean and Variance, Different Distribution)



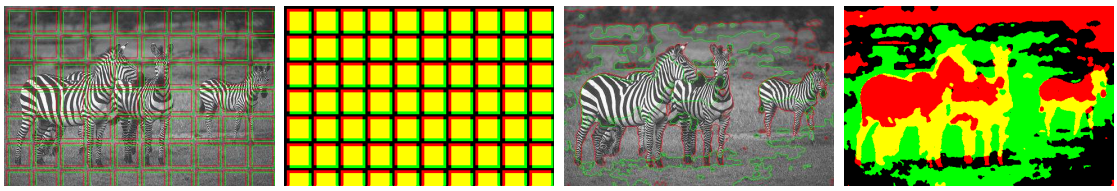
(a) (b) (c) (d)

Figure 3-3: Artificial 4-ary Segmentation



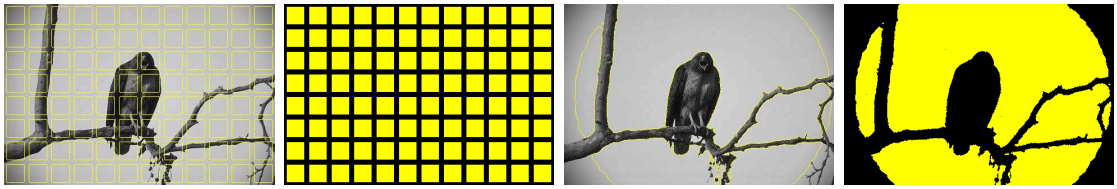
(a) (b) (c) (d)

Figure 3-4: Binary Segmentation of Zebra



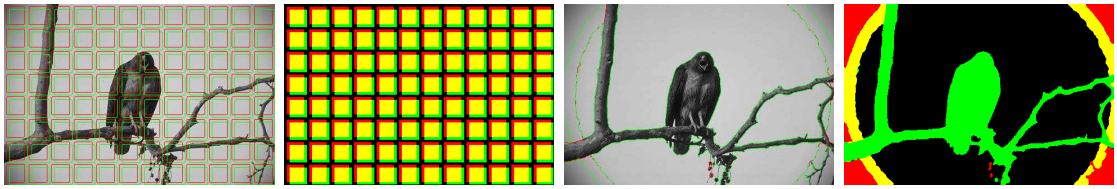
(a) (b) (c) (d)

Figure 3-5: 4-ary Segmentation of Zebra



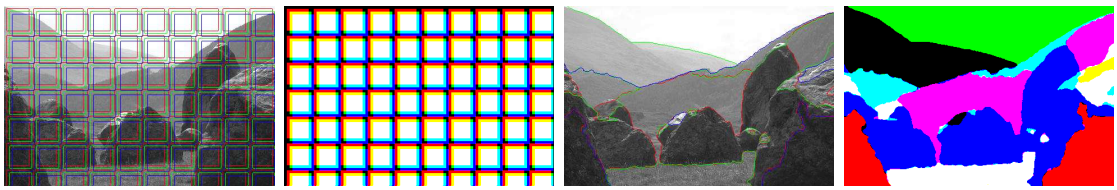
(a) (b) (c) (d)

Figure 3-6: Binary Segmentation of Bird



(a) (b) (c) (d)

Figure 3-7: 4-ary Segmentation of Bird



(a) (b) (c) (d)

Figure 3-8: 8-ary Segmentation of Scenery

3.3 Comments

The results in the previous section show both the strengths and weaknesses of the algorithm. In the artificially generated images, the segmentation was done perfectly. What was most surprising is that in Figure 3-2, where the regions can hardly be distinguished by a human, the algorithm was able to segment the image. It also does fairly well in some of the real images. In the binary segmentation of the zebra image in Figure 3-4, the entire zebra is grouped as one region instead of separating each stripe. This achievement is something that not many algorithms are able to boast. The results from the Berkeley Dataset [16] show that almost all the current algorithms segment the stripes of the zebra.

However, there are definite places for improvement in the algorithm. Images with smoothly varying colors from illumination gradients are not segmented very well. This effect can be seen in both the zebras in Figure 3-5 and the background in Figures 3-6 and 3-7. The final segmentation, although not the ground truth, does fit with the minimization of the energy functional. The main reason why these segmentations are not ideal is because the information theoretic term in the energy functional is based solely on a single pixel intensity; it does not take advantage of the inherent structure of the region around each pixel.

The extreme case of this flaw occurs with an image composed solely of textures. For example, Figure 3-9 shows an image that clearly can be segmented by a human, but will fail using this algorithm. The PDFs of the two regions are exactly the same, but the orientation of the structural texture is different. This implies that a robust algorithm should be based on some sort of texture and orientation measure instead of just pixel intensity.

Another problem with the current implementation of this algorithm is the restriction on the number of regions in an M-ary segmentation. M must be an integer power of 2, which means that images with the number of regions being a non-integer powers of 2 will most likely not be segmented properly. Artificial regions will sometimes surface. For example, the fuchsia and aqua colored regions are both from the

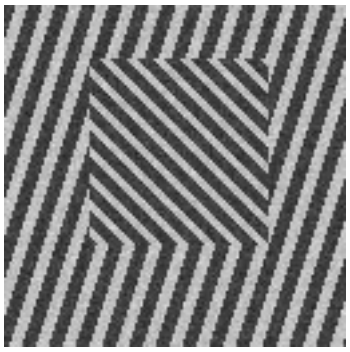


Figure 3-9: Textured Image

same mountain and have very similar PDFs. However, because the segmentation required 8 regions, the separation occurred because of a slight change in PDF that is indistinguishable by the eye.

The algorithm that this thesis focuses on will attempt to solve these problems to improve the image segmentation algorithm.

Chapter 4

The Improved Algorithm

The previous chapter mentioned some problems of the algorithm proposed in [13] and improvements that could be made. In this chapter, details about the new algorithm will be explicitly stated, and preliminary results will be presented.

4.1 The Initial Algorithm

The basis of the improvements in the algorithm are to incorporate a texture measure into the energy functional. This algorithm is a very simple extension from the one published. The main difference is that we will be using a vector image instead of a scalar image. The vector image is generated from the outputs of the Steerable Pyramid, which outputs images at R orientations and S scales. It combines these RS images with the original image and the lowest resolution image to form a vector image of dimension $d = RS+2$. We then proceed with the previous formulation, with the minor difference that each pixel is now a vector instead of a scalar. The energy functional in Equation 3.1, and the velocity vector in Equation 3.3 just become the following

$$E(\mathbf{c}) = -|\Omega|\hat{I}(G(\mathbf{c}); L_C(\mathbf{c})) + \alpha \oint_C ds \qquad \forall \mathbf{c} \in \mathcal{C} \tag{4.1}$$

$$\overrightarrow{V}(\mathbf{c}) = \left[log\frac{\hat{p}_+(G(\mathbf{c}))}{\hat{p}_-(G(\mathbf{c}))} - \alpha\kappa \right] \overrightarrow{N} \qquad \forall \mathbf{c} \in \mathcal{C} \tag{4.2}$$

Thus, the only real difference between the old and new algorithm is that the vector-

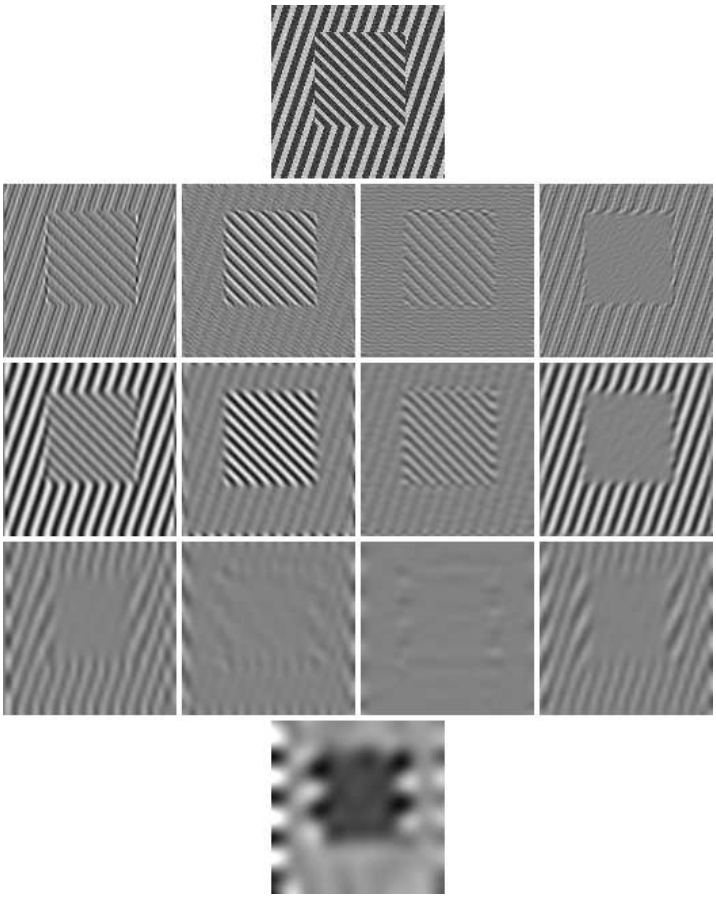


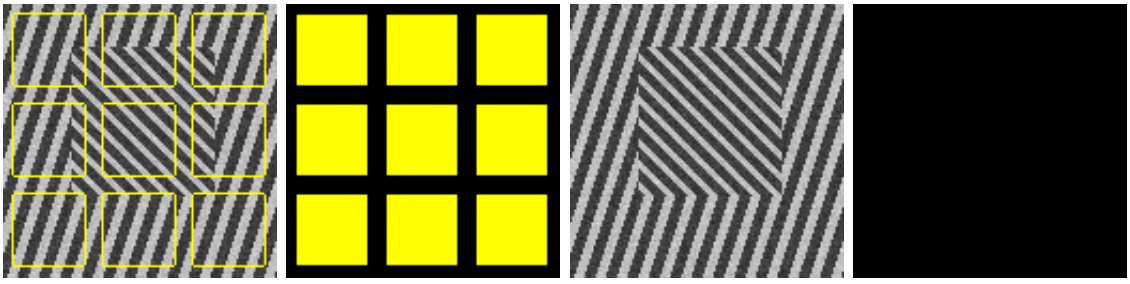
Figure 4-1: Steerable Pyramid Output on Textured Image

ized image needs to be formed from the Steerable Pyramid, and a multi-dimensional PDF needs to be estimated.

4.2 Initial Results

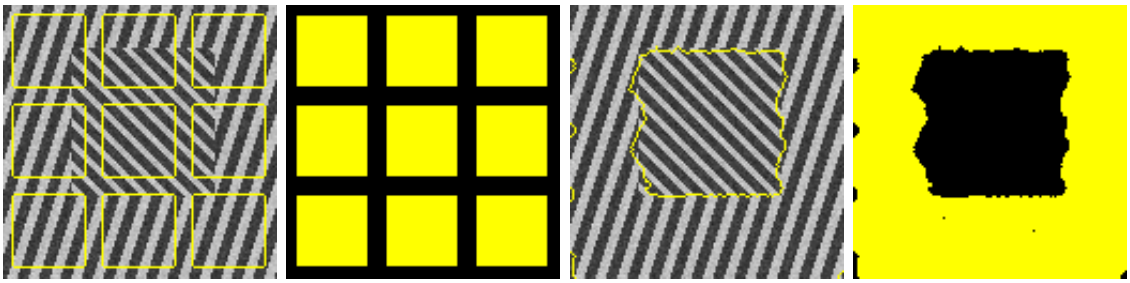
This new algorithm was implemented to see if the concept would work. The Steerable Pyramid was first computed at 4 orientations and 3 scales on the image in Figure 3-9. The resulting vectorized image was composed of the images in Figure 4-1.

The segmentation result using the algorithm from [13] is shown in Figure 4-2, and the result with the proposed algorithm is shown in Figure 4-3. In the scalar segmentation, the result is useless; because the PDFs of the two regions are exactly the same, the segmentation groups the entire image as one region. However, when



(a) (b) (c) (d)

Figure 4-2: Artificial Binary Scalar Segmentation on Textured Image



(a) (b) (c) (d)

Figure 4-3: Artificial Binary Vector Segmentation on Textured Image

applying the texture measure of Steerable Pyramids with the segmentation, the algorithm performed much better. The rough edges on the object are due partially to a discretization problem and to the low resolution images appearing very block-like at higher resolutions. Although the result is not perfect, and only one test image is shown, it shows that there is potential in improving the algorithm to support textured images for better segmentation.

4.3 Proposed Work

Although the initial algorithm does show a proof of concept, a lot of work needs to be done in improving it. A 128x128 grayscale image takes about 3 seconds to segment under the scalar algorithm. However, a 128x128 grayscale image outputted at 4 orientations and 3 scales takes over 30 minutes to segment under the vector algorithm. The main difference between the two cases is the 18-dimensional KDE. As stated in [21, p.30], the IFGT is proportional to a polynomial order of the dimension (i.e. d^p),

where p is a function of the number of terms needed in the Taylor Series expansion to achieve the desired error. From the test image in Figure 4-3, the typical p is around 5. This means that for the increase of dimension from 1 to 18, the dependence on the dimension changes from $\mathcal{O}(1)$ to $\mathcal{O}(18^5)$. This increase in complexity explains why the vectorized image takes so much longer to compute.

The initial segmentation was run on such a small image because larger images contain more scales (increased vector dimension), and more pixels. The complexity of the IFGT is asymptotically equal to $\mathcal{O}(d^p(M+N))$, where M is the number of target points and N is the number of source points. Increasing from a 128x128 image to a 256x256 image quadruples both M and N in addition to increasing d. Segmenting anything but very small images is too computationally intensive to be completed, and approximations need to be made to decrease the complexity. Therefore, the approach to improve this algorithm will incorporate the following ideas:

1. Reducing dimensionality of the vector image
2. Product kernel approximation to the fully joint estimate
3. Multi-leveled binary segmentations to achieve an M-ary segmentation

4.3.1 Reducing Dimensionality

Trying to reduce the dimensionality of the vectorized image is equivalent to deciding which of the oriented and scaled outputs is least significant in reconstructing the image. Although this clearly has advantages in computational efficiency, it is also a very interesting philosophical question. It would be very interesting to compare the most significant images to what humans perceive as a way to try and understand the ever-complicated human visual system. Keeping these thoughts aside for now, the methodology of reducing the dimensionality will be explored a little.

From Equation 2.18, we know that the original image can be reconstructed as a sum of linear transformations of the pyramid outputs. For convenience, the synthesis equation is replicated here.

$$\mathbf{x} = \sum_{i=0}^{d-1} \Theta_i \mathbf{y}_i$$

We can define a binary vector, $\mathbf{u} \in \{0,1\}^d$, where each component of $\mathbf{u}$, u_i, determines whether or not the corresponding output image, $\mathbf{y}_i$, is used in the reconstruction. Thus, an estimate for the synthesis stage can be made by the following

$$\hat{\mathbf{x}} = \sum_{i=0}^{d-1} \Theta_i \mathbf{y}_i u_i \tag{4.3}$$

Then, we can define an error function, $e(\hat{\mathbf{x}}, \mathbf{x})$, which evaluates how close of a reconstruction $\hat{\mathbf{x}}$ is to $\mathbf{x}$. One possible error function is just the square of the L2 norm.

$$e(\hat{\mathbf{x}}, \mathbf{x}) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2^2 = (\hat{\mathbf{x}} - \mathbf{x})^T(\hat{\mathbf{x}} - \mathbf{x}) \tag{4.4}$$

Our goal is to minimize the error function while also minimizing the number of images used. In other words, we want to perform the following optimization

$$\mathbf{u}^* = \arg \min_{\mathbf{u} \in \{0,1\}^d} \left[(1-\beta) e(\hat{\mathbf{x}}, \mathbf{x}) + \beta \sum_{i=0}^{d-1} u_i \right] \tag{4.5}$$

This optimization is similar to the ones posed in [8, 25], and can most likely be solved in a similar fashion.

4.3.2 Product Kernel

Probably the most straightforward approximation to speed up the KDE is to estimate the PDF with a product kernel. The current implementation (Equation 2.14) of the KDE uses a jointly Gaussian kernel. The kernel is replicated here for convenience.

$$K^G(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\|\mathbf{x}\|^2}$$

The idea of a product kernel is simple; split this kernel into the product of scalar kernels. Thus, if $\dim(\mathbf{x}) = d$, we can rewrite the product kernel as

$$K_p(\mathbf{x}) = \prod_{i=0}^{d-1} K(x_i) \tag{4.6}$$

If we are still using the Gaussian kernel, this product kernel just becomes

$$K_p^G(\mathbf{x}) = \prod_{i=0}^{d-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x_i^2} \tag{4.7}$$

We can simplify this kernel to be just

$$\begin{aligned} K_p^G(\mathbf{x}) &= \frac{1}{(2\pi)^{d/2}} \prod_{i=0}^{d-1} e^{-\frac{1}{2}x_i^2} \\ &= \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}x_0^2 - \cdots - \frac{1}{2}x_{d-1}^2} \\ &= \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\sum_{i=0}^{d-1} x_i^2} \\ &= \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\|\mathbf{x}\|_2^2} \\ &= \frac{1}{(2\pi)^{d/2}} e^{-\frac{1}{2}\left(\sqrt{\|\mathbf{x}\|_2^2}\right)^2} \\ &= \frac{1}{(2\pi)^{(d-1)/2}} K^G\left(\sqrt{\|\mathbf{x}\|_2^2}\right) \end{aligned} \tag{4.8}$$

This approximation basically maps the IFGT of the original source points in d-dimensions to a 1-dimensional estimate. Therefore, the computation complexity is reduced from $\mathcal{O}(d^p(M+N))$ to $\mathcal{O}(d+M+N)$. In addition, it will allow the KDE to use a different bandwidth for each output image which is desired because each output image has a different sample variance.

4.3.3 Multi-leveled Binary Segmentations

The last improvement that will be incorporated into this thesis is the M-ary segmentation problem. As stated previously, the current implementation of the segmentation algorithm only allows problems where $M = 2^K$ where $K \in \mathbb{Z}$. The rough outline of a possible method to fix this is as follows:

1. Compute a regular binary segmentation resulting in R_0^0 and R_0^1, where R_i^j is the ith region for the jth iteration
2. Automatically decide if any of the new regions should be further segmented

(assume R_i^j needs to be segmented again)

3. Run a binary segmentation on all regions that need further segmentation resulting in R_{2i}^{j+1} and R_{2i+1}^{j+1} while constricting the support of the new regions to the support of R_i^j
4. Repeat from 2 until no more regions need to be further segmented

Though this procedure seems straightforward, the autonomous decision of whether a region should be split into more regions is actually quite difficult. One initial idea that has not been implemented or tested is to just run the binary segmentation on each region in step 2. If the algorithm segments the region into two new ones, then the original region should have been further segmented into exactly what the algorithm found. If the algorithm does not segment the original region into more regions, this could indicate that no further segmentation is needed. This idea is not necessarily the best method, but it may be one of the least complicated possible solutions. Other methods of deciding whether one region is composed of more need to be further researched.

Chapter 5

Timeline

I plan to finish my research and implementations for this thesis by February 2009. I will spend the remainder of the 2009 spring semester writing my thesis and finishing any minor simulations needed. My timeline is as follows:

- June 2008 - Product Kernel
  - Implement the product kernel improvement
  - Work on subpixel accurate level set methods
  - Improve the GUI for the program
- July 2008 - Dimensionality Reduction
  - Research methods to solve the optimization in the dimensionality reduction problem
  - Implement the solution to the optimization
- August-September 2008
  - Implement the M-ary segmentation method presented
  - Research other methods to decide if a region should be split
  - Research other methods for M-ary segmentations using level set methods
- October 2008

  - Implement the Steerable Pyramids in C++
  - Test varying filters, scales, and orientations

- November-December 2008

  - Test varying parameters for the algorithm
  - Run simulations on Berkeley's Image Database [16]

- January 2009

  - Consider possible extensions with smoothly varying textures
  - Test code on multiples operating systems
  - Clean up and package code
  - Test code on multiples operating systems

- February-May 2009

  - Finish minor details and simulations
  - Write Master's Thesis

Bibliography

[1] D. Adalsteinsson and J. Sethian. The fast construction of extension velocities in level set methods. *Journal of Computational Physics*, 148:2–22, 1998.

[2] M. Bertero, T.A. Poggio, and V. Torre. Ill-posed problems in early vision. *Proceedings of the IEEE*, 76(8):869–889, Aug 1988.

[3] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, 1986.

[4] T.F. Chan and L.A. Vese. An efficient variational multiphase motion for the mumford-shah segmentation model. *Signals, Systems and Computers, 2000. Conference Record of the Thirty-Fourth Asilomar Conference on*, 1:490–494 vol.1, 2000.

[5] T.F. Chan and L.A. Vese. Active contours without edges. *Image Processing, IEEE Transactions on*, 10(2):266–277, Feb 2001.

[6] Yining Deng and B. S. Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(8):800–810, 2001.

[7] Piotr Dollar, Zhuowen Tu, and Serge Belongie. Supervised learning of edges and object boundaries. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1964–1971, Washington, DC, USA, 2006. IEEE Computer Society.

[8] M. Elad. Why simple shrinkage is still relevant for redundant representations? *Information Theory, IEEE Transactions on*, 52(12):5559–5569, Dec. 2006.

[9] W. T. Freeman and E. H. Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 13(9):891–906, 1991.

[10] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[11] M. Grayson. The heat equation shrinks embedded plane curves to round points. *Journal of Differential Geometry*, 26(2):285–314, 1987.

[12] Leslie Greengard and John Strain. The fast gauss transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[13] Junmo Kim, J.W. Fisher III, A. Yezzi, M. Cetin, and A.S. Willsky. A nonparametric statistical method for image segmentation using information theory and curve evolution. *Image Processing, IEEE Transactions on*, 14(10):1486–1502, Oct. 2005.

[14] R. Malladi, J.A. Sethian, and B.C. Vemuri. Shape modeling with front propagation: a level set approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(2):158–175, Feb 1995.

[15] J. Marroquin, S. Mitter, and T. Poggio. Probabilistic solution of ill-posed problems in computational vision. *Journal of the American Statistical Association*, 82(397):76–89, March 1987.

[16] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[17] David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):530–549, 2004.

[18] Stanley Osher and Ronald Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, 31 October 2002.

[19] Nikos Paragios and Rachid Deriche. Geodesic active regions and level set methods for supervised texture segmentation. *Int. J. Comput. Vision*, 46(3):223–247, 2002.

[20] E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, 1962.

[21] Vikas Chandrakant Raykar. *Scalable Machine Learning for Massive Datasets: Fast Summation Algorithms*. PhD dissertation, University of Maryland, Department of Computer Science, 2007.

[22] J.A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, second edition, 13 June 1999.

[23] Eero P. Simoncelli and William T. Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *International Conference on Image Processing*, volume 3, pages 444–447, 23-26 Oct. 1995, Washington, DC, USA, 1995.

[24] Eero P. Simoncelli, William T. Freeman, Edward H. Adelson, and David J. Heeger. Shiftable multi-scale transforms. *IEEE transactions on informations theory*, 38(2), 1992.

[25] R. Tibshirani. Regression shrinkage and selection via the lasso. Technical report, University of Toronto, June 1994.

[26] J. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[27] Yong-Gang Wang, Jie Yang, and Yu-Chou Chang. Color-texture image segmentation by integrating directional operators into jseg method. *Pattern Recogn. Lett.*, 27(16):1983–1990, 2006.

[28] C. Yang, R. Duraiswami, N. Gumerov, and L. Davis. Improved fast gauss transform and efficient kernel density estimation. *IEEE International Conference on Computer Vision*, pages 464–471, 2003.