

UNIVERSITY OF CALIFORNIA

SANTA CRUZ

**Piecewise-Smooth Modeling of Digital Images**

A thesis submitted in partial fulfillment of the  
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

**Paul J. Ausbeck, Jr.**

June 1996

The Dissertation of Paul J. Ausbeck, Jr.  
is approved:

---

Professor Glen G. Langdon, Jr., Chair

---

Professor Pat Mantey

---

Professor Richard Hughey

---

Dean of Graduate Studies



## Table of Contents

1. Introduction .....	1
2. Canonical Piecewise Smooth Example .....	3
2.1 Model Extraction .....	4
2.2 Model Coding .....	7
2.3 Coding Experiments.....	12
3. Moments of Two Dimensional Domains.....	13
3.1 A Digression .....	13
3.2 Moments.....	13
3.3 Domain Operations.....	15
3.4 The Two Dimensional Linear System.....	15
3.5 Second and Third Order Systems .....	16
3.6 Solving Least Squares Systems .....	17
3.6.1 Closed Form Solution for the Linear Problem .....	17
3.6.2 Cholesky Factorization of Symmetric Positive Semidefinite Systems .....	18
3.7 Error Function .....	26
3.8 Space/Precision Issues.....	27
3.9 Summary.....	28
4. Domain Extraction .....	29
4.1 Motivation .....	29
4.2 Background .....	29
4.3 Greedy Domain Growing .....	31
4.4 Implementation .....	32
4.4.1 Concepts.....	32
4.4.2 Algorithms .....	34
4.5 Proof of Correctness .....	39
4.5.1 Definitions .....	39
4.5.2 Overview .....	39
4.5.3 Details .....	39
4.6 Algorithmic Complexity .....	41
4.6.1 Dynamic Size Limiting.....	43
4.7 Summary.....	43
5. Domain Smoothing .....	45
5.1 Motivation .....	45
5.2 Background .....	46
5.3 Boundary State Machine.....	46
5.4 Rasterization.....	47
5.5 Boundary Cost Function .....	48
5.6 Smoothing Transformations.....	49

5.6.1 Move Sets .....	50
5.6.2 Limiting Move Depth .....	51
5.7 Determining $\Delta b$ .....	52
5.8 Determining $\Delta l$ .....	53
5.8.1 Diagonal Separators .....	54
5.8.2 Boundary Length Accounting .....	55
5.9 Convergence .....	56
5.9.1 Interfering Domains.....	56
5.9.2 Subtractive Interference .....	57
5.9.3 Summary of Convergence Requirements.....	58
5.10 Correctness .....	58
5.11 Computational Complexity.....	59
5.12 Limitations .....	60
5.13 Experiments.....	60
5.14 Summary.....	65
6. Domain Boundary Coding .....	67
6.1 Background.....	67
6.1.1 Chain Coding.....	67
6.1.2 Raster Neighborhood Coding.....	68
6.2 A Boundary Partition Classification .....	69
6.3 Chain Coding via Strokes .....	70
6.3.1 Binary Decisions .....	71
6.3.2 Stroke Start Points.....	72
6.3.3 Stroke Chain Termination.....	74
6.3.4 Stroke Chains.....	75
6.4 Experiments.....	76
6.5 Summary.....	80
7. Geometry Implicit Coding of Two Dimensional Polynomials.....	81
7.1 Background.....	81
7.2 Sentinel Points and Polynomial Reconstruction .....	82
7.3 Choosing Sentinel Points.....	83
7.3.1 Zero Order System.....	83
7.3.2 First Order System .....	84
7.3.3 Second Order System .....	86
7.3.4 Third Order System .....	89
7.3.5 Examples.....	91
7.4 Sentinel Points of Under-Constrained Domains .....	93
7.5 Quantization .....	94
7.5.1 Variable Quantization.....	97
7.6 Optimizing Quantized Sentinel Point Values.....	99
7.7 Coding.....	99
7.7.1 Quantization Model .....	100
7.7.2 Prediction.....	100
7.7.3 Experiments.....	101
8. Coding Experiments.....	103

8.1	Generating Piecewise-Smooth Image Models.....	103
8.1.1	Coder Performance.....	106
8.2	Coding the Image Model.....	107
8.2.1	Decoder Performance.....	109
8.3	Data.....	109
8.4	Sample Partitions.....	111
8.5	JPEG Comparison.....	114
8.6	Summary.....	116
9.	Conclusion.....	117
9.1	Main Contributions.....	117
9.2	Future Work.....	118
9.2.1	Model Extraction.....	119
9.2.2	Model Coding.....	119
9.2.3	Extensions.....	120
10.	References.....	121

## List of Figures

Figure 2.1 Synthetic Image Syn15.....	3
Figure 2.2 Domains Extracted from Syn15.....	5
Figure 2.3 Synthetic Image Syn16.....	6
Figure 2.4 Initial Domains Extracted from Syn16 .....	6
Figure 2.5 Syn16 Domain Extraction with Noise Suppression .....	7
Figure 2.6 Smoothed Syn16 Domains .....	7
Figure 2.7 Implicit Code Points .....	9
Figure 2.8 Decoder Reconstruction, $Q_s = 64$ .....	9
Figure 3.1 Zero Propagation .....	22
Figure 3.2 Sample Domains.....	23
Figure 4.1 Traveling a Domain Boundary.....	35
Figure 4.2 Hole Discovery .....	36
Figure 4.3 Greedy Domain Extraction Algorithm .....	38
Figure 5.1 Irregular Boundary .....	45
Figure 5.2 Move Sets.....	51
Figure 5.3 Perceived Boundary Length .....	54
Figure 5.4 Boundary Length Accounting.....	55
Figure 5.5 Interfering Domains .....	57
Figure 5.6 Connection Hulls.....	59
Figure 5.7 Synthetic Image Syn4.....	62
Figure 5.8 Four Domain Partition of Syn4.....	62
Figure 5.9 Smoothed Syn4, $\alpha = 256$ .....	63
Figure 5.10 Smoothed Syn4, $\alpha = 512$ .....	63
Figure 5.11 Smoothed Syn4, $\alpha = 1024$ .....	63
Figure 5.12 Smoothed Syn4, $\alpha = 8192$ .....	63
Figure 5.13 Smoothed Syn4, $\alpha = 100000$ .....	64
Figure 5.14 Smoothed Syn4, Restorative mode, $\alpha = 1$ .....	64
Figure 5.15 Unsmoothed 100 Domain Lena.....	65
Figure 5.16 Smoothed 100 Domain Lena, $\alpha = 1024$ .....	65
Figure 6.1 Stroke Example.....	70
Figure 6.2 Boundary and Stroke Start Points 100 Domain Lena .....	74
Figure 6.3 Boundary and Stroke Start Points 3200 Domain Lena .....	74
Figure 6.4 Stroke Code vs Q-Code Smoothed and Unsmoothed Lena.....	80
Figure 7.1 Zero Order Sentinel Points for Syn15.....	91
Figure 7.2 First Order Sentinel Points for Syn15 .....	91
Figure 7.3 Second Order Sentinel Points for Syn15 .....	92
Figure 7.4 Third Order Sentinel Points for Syn15 .....	92

Figure 7.5 Third Order Sentinel Points for a 100 Domain Lena.....	93
Figure 7.6 Third Order Sentinel Points for an 800 Domain Lena .....	93
Figure 7.7 Quantization Noise Uniform vs 3 <sup>rd</sup> Order Lena.....	97
Figure 8.1 Decoder Functional Diagram .....	108
Figure 8.2 Domain Boundaries 200 Domain Lena .....	112
Figure 8.3 Domain Boundaries 100 Domain Cameraman .....	112
Figure 8.4 Domain Boundaries 400 Domain Cameraman .....	113
Figure 8.5 Domain Boundaries 400 Domain Baboon.....	113
Figure 8.6 Domain Boundaries 100 Domain Miss America .....	113
Figure 8.7 Domain Boundaries 200 Domain Miss America .....	114
Figure 8.8 Rate Distortion Comparison Miss America .....	115
Figure 8.9 Rate Distortion Comparison Lena.....	115
Figure 8.10 Rate Distortion Comparison Cameraman.....	115
Figure 8.11 Rate Distortion Comparison Baboon .....	115

# Piecewise-Smooth Modeling of Digital Images

*Paul J. Ausbeck Jr.*

## Abstract

We introduce the piecewise-smooth image model. We develop efficient algorithms for model extraction and representation when the smooth model component is limited to third order two-dimensional polynomials. We apply the model to lossy image coding and compare its rate-distortion performance against the JPEG image compression standard. On some images tested, the piecewise-smooth code outperforms JPEG by a wide margin, and it is never significantly inferior.

As byproducts, we introduce moment operators to aid in multidimensional piecewise-smooth surface fitting. We extend the Cholesky factorization of symmetric positive definite matrices to symmetric positive semidefinite matrices. We develop a general image segmentation algorithm that outperforms any previously reported method. We introduce the raster-break measure of boundary noise and apply it to a fast state-machine boundary smoother. We develop the stroke method for chain coding contours. It is the first method to efficiently solve the three-direction chain termination problem. We develop the method of sentinel points for minimally coding polynomial surfaces over arbitrary two-dimensional domains.



## Acknowledgments

The author wishes to thank his advisor, Glen G. Langdon, Jr., for his enthusiastic support and his direction in finding a dissertation topic.

## 1. Introduction

*Despite considerable efforts over a long period, the theory and practice of segmentation remained primitive for two reasons. First, it was well-nigh impossible to formulate precisely in terms of the image or even of the physical world what the exact goals of segmentation were. ...David Marr, Vision (1982).<sup>1</sup>*

In introducing his idea of the  $2\frac{1}{2}$ -dimensional sketch, Marr presented a simple image of two leaves that gave vision to his belief that the idea of segmentation was bound to fail. We won't reproduce that image here, but in the next chapter we introduce a synthetic benchmark problem with identical characteristics. That benchmark is the real introduction to the *piecewise-smooth image model*.

For now we will be content to say that the piecewise-smooth image model lays a mathematical foundation for the image segmentation problem. Because of this, we use different segmentation terms than those with which the reader may already be familiar. We refer to a segment or region as a *domain*. We refer to the segmentation process as *model* or *domain extraction* and to the resulting segmentation as a an *image partition*. We apologize in advance if by Chapter 9 the reader does not appreciate the use of these terms.

Image segmentation is really only the lowest level attack on the problem of *image understanding*. The fundamental importance of the piecewise-smooth model to image understanding can be most easily understood by relating it to the problem of realistic image synthesis. All of visual cues\* that allow us to give life to a synthetic scene are either piecewise-smooth intensity processes or texture processes linearly interacting with an underlying piecewise-smooth model. The example presented in Chapter 1 clearly shows how even simple smooth intensity processes give depth and character to an image.

The piecewise-smooth image model canonically represents two of the most important perceptual elements of an image: smoothly varying intensity patches and step intensity discontinuities. Further, if the piecewise-smooth image component is captured and isolated, the remaining small-scale or texture component can be more easily analyzed.

In Chapter 2 we develop a simple problem example as an overview of the piecewise-smooth image model. The moment operators of Chapter 3 enable the greedy domain-growing model extraction algorithm of Chapter 4. In Chapter 5 we define the raster-break as a formal measure of boundary noise and use it to develop a state-machine boundary smoother. In Chapter 6 we begin to address model representation by developing the stroke domain boundary code. In Chapter 7 we develop the sentinel point method for minimally encoding polynomials over arbitrary two-dimensional domains. In Chapter 8 we apply the piecewise-smooth machinery to lossy image coding and compare the results to the JPEG image compression standard. Chapter 9 is a summary of our results in more detail than presented here.

---

\* An arguable exception is shape from texture.

## 2. Canonical Piecewise Smooth Example

In introducing a new subject, a canonical problem example is often most useful. Figure 2.1 is a synthetic example of a piecewise smooth gray scale image. It has 256x256 pixels, each with gray level value between 0 and 255 and being the fifteenth in a series of experiments is designated Syn15. The image was generated with the 11 pixel intensity functions in Table 2.1. Domain **A** was split into three pieces by the subsequent overlay of domains **B**, **E**, **F**, and **K**, resulting in a total of 13 connected domains.

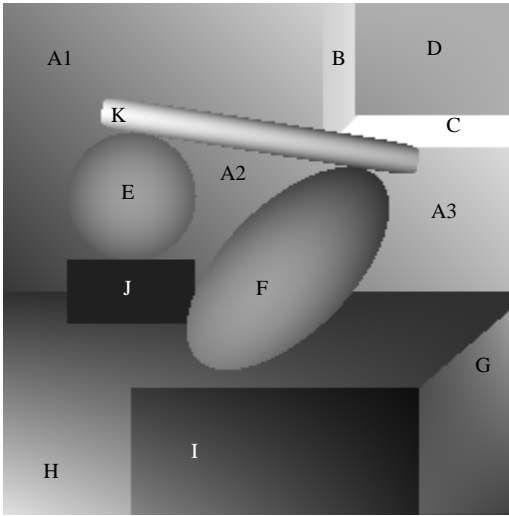


Figure 2.1  
Synthetic Image Syn15

A	$.5x - .5y + 128$
B	$x + 48$
C	255
D	$.0060xy - 1.52y + 176$
E	$-.062x^2 - .062y^2 + 8x + 12y - 672$
F	$-.039x^2 - .039y^2 - .0469xy + 16.8x + 17.4y - 2189$
G	$-.027xy + 6.23x + 5.93y - 1276$
H	$-.00625xy + .9x + 1.6y - 182$
I	$-.00625xy + .9x + 1.6x - 230$
J	32
K	$-.048x^2 - 1.94y^2 + .61xy - 28.5x + 178y - 3798$

Table 2.1  
Synthetic Patch Intensity Functions

In a piecewise smooth image, each of the pieces, or domains, contains a subset of the pixels of the image. The union of all such domains contains all the pixels of the image and the intersection of any two domains is null. Each domain is also connected. That is, for any two pixels in a domain:  $\forall p \in D_i, \forall j \forall k (p_j \xrightarrow{\text{rectilinear path within } D_i} p_k)$ . A rectilinear path contains only vertical and horizontal movements through the centers of square pixels that.

Each domain has a smooth pixel intensity function that determines the values of its pixels. Since any image is piecewise smooth if a different domain is assigned to each pixel, to make this classification interesting we must also limit the number of domains. For our purposes, an image is piecewise smooth if it can be efficiently modeled with  $N_D$  domains and  $N_D < \frac{|\mathbb{I}|}{\log_2 |\mathbb{I}|}$ , where  $|\mathbb{I}|$  is the number of pixels in the image.

For the rest of this thesis, we further restrict the pixel intensity functions to two dimensional polynomials of order three or less:

$$\Phi(x, y) = ax^3 + bx^2y + cxy^2 + dy^3 + fx^2 + gxy + hy^2 + ix + jy + k .$$

Syn15 itself is comprised of second order or smaller polynomials as it was produced during development of the second order extraction procedure.

## 2.1 Model Extraction

It seems clear that to minimally code Syn15 it would be desirable to discover the original processes used to generate it. In this case, these piecewise smooth processes are known beforehand so we can quantitatively evaluate any extraction procedure. Fortunately (for a thesis topic), a literature search<sup>2,3,7,10,11,12</sup> does not provide a technique that seems appropriate for recovering all 13 domains of Syn15. Its intensity function contains too much gradient that is not associated with a boundary between domains. Additionally, several edges in the image do not have any intensity discontinuity. Taken together, these two characteristics make the recovery of an accurate partition an impossible task for previously reported segmentation algorithms: edge linking, split and merge, thresholding, or region growing. Existing segmentation techniques are discussed further in Chapter 4.

In Chapter 3 we develop techniques that use moments to help work with polynomial approximation functions over arbitrary two-dimensional domains. In particular, we develop  $O(1)$  (bounded by a constant) methods for the following domain operations:

- Determining a domain's approximate pixel intensity function.
- Determining the total squared error between a domain's approximation and the actual pixel values in the domain.
- Merging two domains.
- Excising one domain from another.
- Adding and removing a pixel from a domain.

With these in hand, in Chapter 3 we develop an algorithm for *greedy domain growing* that is particularly effective for partitioning images such as Syn15. Figure 2.2 shows the domain boundaries extracted from Syn15 by merging until only 13 domains remain. Table 2.2 shows the corresponding extracted intensity functions. The domain alignment is within a single raster row or column for all domains, and the mean squared error (MSE) between the pixel values predicted by the extracted intensity functions and their actual values is 0.228.

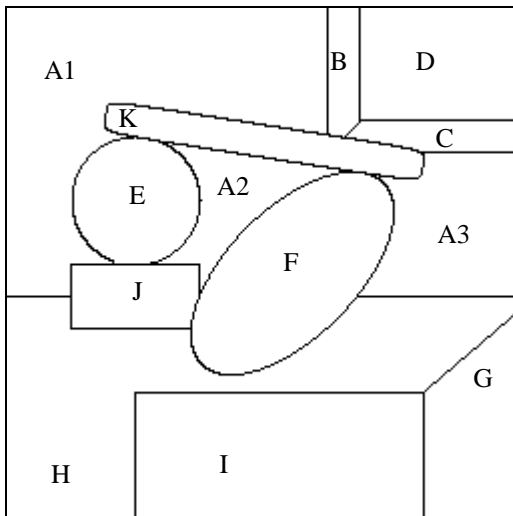


Figure 2.2  
Domains Extracted from Syn15

A1	$.5x - .5y + 127.75$
B	$x + 48$
C	255
D	$.0060xy - 1.53y + .013x + 174.4$
E	$-.062x^2 - .062y^2 + 8x + 12y - 672.6$
F	$-.039x^2 - .039y^2 - .0469xy + 16.8x + 17.4y - 2189$
G	$-.007y^2 - .027xy + 6.3x + 6.28y - 1320$
H	$-.0063xy + .9x + 1.6y - 182.6$
I	$-.0062xy + .9x + 1.6x - 231.3$
J	32
K	$-.048x^2 - 1.94y^2 + .61xy - 28.5x + 178y - 3796.2$

Table 2.2  
Pixel Intensity Functions Extracted from Syn15

Since few real images are perfectly piecewise smooth, it is instructive to examine how the algorithm behaves against a member of the larger class of images that are *piecewise smoothly textured*. Synthetic image Syn16, shown in Figure 2.3, is Syn15 with additive gaussian noise ( $\sigma=16$ ). The result of partitioning Syn16 via greedy domain is shown in Figure 2.4. One of several obvious problems is that domains **B** and **C** have been merged.

The algorithm of Chapter 3 is able to jointly optimize both a bulk property such as MSE and a boundary property such as overall length. Boundary length is important from a coding perspective in that longer boundaries take more bits to encode. A generalization of this coding cost argument, called the Minimum Descriptive Length Principle<sup>8</sup>, asserts that if two or more possible interpretations of a set of data exist, *the simplest is often the best*. Another way of saying this is that *nature prefers simplicity*. It certainly would have been quite difficult to generate a synthetic image with the boundaries of Figure 2.4.

The result of performing a joint optimization of boundary length and MSE with a boundary length weighting factor of 256 is shown in Figure 2.5. The result contains the correct number of domains and has good boundary alignment everywhere where the signal-to-noise ratio is greater than -6 dB.

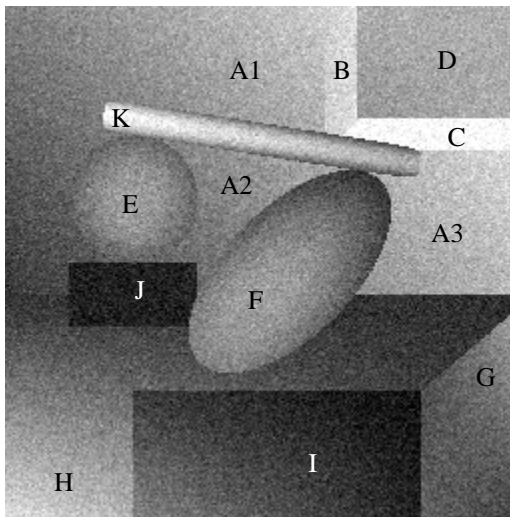


Figure 2.3  
Synthetic Image Syn16

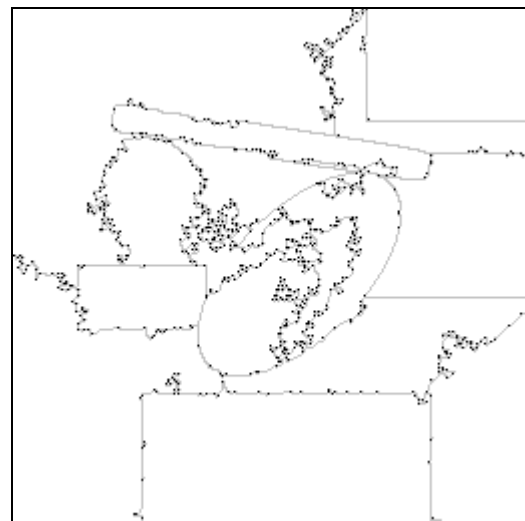


Figure 2.4  
Initial Domains Extracted from Syn16

Although the partition of Figure 2.5 is quite good, much of the boundary is perceived as noisy or irregular. In addition to being perceptually somewhat annoying, this noisy boundary is unlikely to correspond to real image features and is also difficult to code compactly. In Chapter 4 we develop a formal measure of boundary noise and use it to develop a procedure to smooth previously extracted boundaries. The core of this procedure is a boundary following state machine that recognizes loci of irregularity and transforms them to smoother

configurations. The method of Chapter 3 allows the smoother to efficiently optimize boundary noise against pixel intensity function error.

Figure 2.6 is the result of smoothing the domains of Figure 2.5. The loci of irregularity, or *raster breaks* in the terminology of Chapter 4, are shown as darker dots along the boundary on both figures. In its present form, the smoother is constrained to monotonically decrease the overall boundary length. Lifting this restriction could result in some increased recovery of the curvature of the circular boundary, but the noise is larger than the underlying signal over much of this boundary and it is not clear exactly what improvement can be attained. Interestingly, the perceived sphere is quite clear even though the noise completely obscures the actual boundary. This is probably due to the parabolic illumination function centered over the circle. This illumination is responsible for turning the circle into a perceived sphere and the persistence of this illusion is quite strong even when the boundary is obscured by noise.

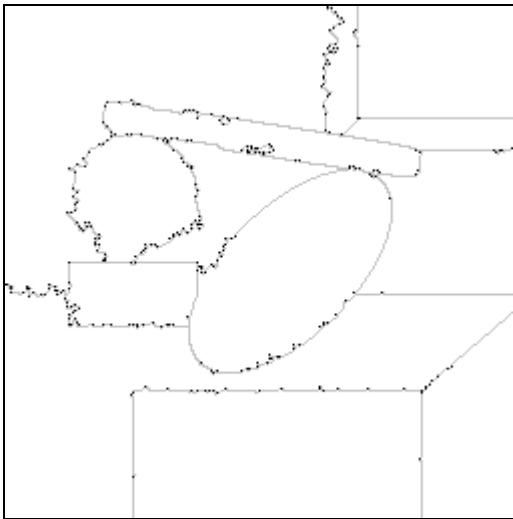


Figure 2.5  
Syn16 Domain Extraction with Noise Suppression

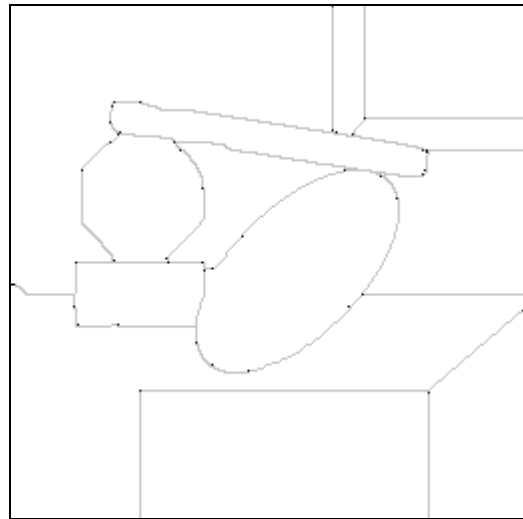


Figure 2.6  
Smoothed Syn16 Domains

## 2.2 Model Coding

A viable piecewise-smooth model extraction procedure allows us to produce some new results for coding model parameters. Chapter 5 describes a chain coding procedure for domain boundaries. A three direction (left, right, straight) chain is used. Further information supplies chain start points and multiplicity: clockwise, counterclockwise or both. The chains are self-



terminating at tee intersections with previously encoded boundaries. Disambiguation information distinguishes terminating and continued corner junctions. Probability estimation contexts are used to further reduce the code length of each chain. Significantly, these contexts are not comprised of a set number of previous events. Rather, they are the boundary states used by the boundary smoother of Chapter 4. The synergy produced by the coder knowing that the boundary is smoothed allows for improved coding performance.

Example	Separators	Raster Breaks	Bits	PSNR
Unsuppressed	4160	904	6020	24.18
Suppressed	2362	348	2895	24.37
Smoothed	1876	48	1208	24.35

Table 2.3  
Syn16 Boundary Coding Statistics

Table 2.3 shows the results of coding the three partitions that we have seen thus far. The first line of the table is for the boundary extracted without noise suppression. The second line corresponds to the noise suppressed extraction. The third line is data for the smoothed boundary. The reduction in code length between lines 1 and 2 is due primarily to a 2:1 reduction in the length of the boundary. This is understandable since the noise suppression was achieved by including boundary length as a parameter of the extraction procedure. Lines 2 and 3 illustrate how reducing the number of raster breaks in the boundary has a dramatic effect on the boundary code string length. The smoothed boundary is only 20% shorter but its code string length has been reduced by 60%.

The last column of Table 2.3 shows how the noise really dominates this problem. When comparing two 256 level grayscale images, the peak signal to noise ratio (PSNR) is defined as

$$20 \log \frac{256}{\sqrt{MSE}}$$

Since Syn16 is Syn15 with  $\sigma=16$  noise, the expected value of PSNR for a

$$\text{perfect extraction of the underlying piecewise smooth model is } 20 \log \frac{256}{16} = 24.08 \text{ dB. All}$$

three examples are very close to this value and the importance of introducing the additional boundary length and smoothness constraints to achieve the final result is clear.

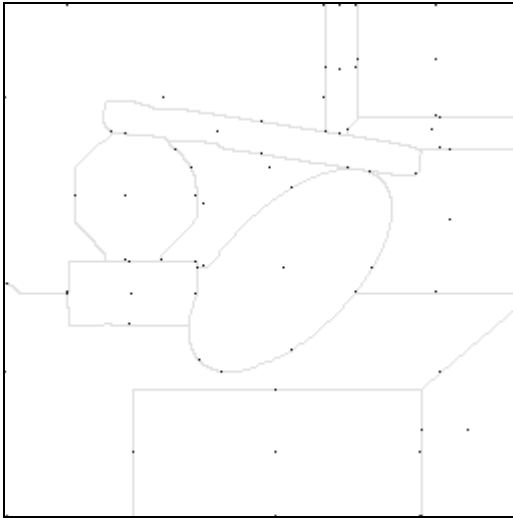


Figure 2.7  
Implicit Code Points

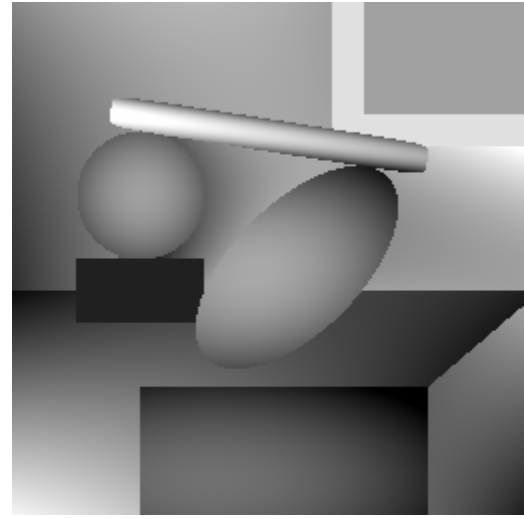


Figure 2.8  
Decoder Reconstruction,  $Q_s = 64$

Chapter 6 presents a new procedure for coding 2D polynomial pixel intensity functions. The key to the method is a set of *sentinel* points, whose locations are calculable only from domain geometry, for which modeled values are transmitted. Using these points, a standard least squares solution is derived for the entire domain. The most important aspect of the procedure is the method for choosing the sentinel points. This becomes particularly interesting for third order polynomials, where 10 points must be chosen for each domain. Figure 2.7 shows the sentinel points for the smoothed model extracted from Syn16. Since this is a second order image, there are six sentinel points per domain.

Since the sentinel points lie within their associated domain, their values are constrained to lie within the range of image representation, or for our 256 level grayscale images, between 0 and 255. The base number of bits necessary to code a sentinel point value is set by an associated quantization step size,  $Q_{step}$ . The quantization step size is the minimum difference between two dissimilar values. Since our values are constrained between 0 and 255, this base bit quantity is  $\log_2 \frac{256}{Q_{step}}$ .

Table 2.4 shows the results of coding Syn15 with no quantization and with eight quantizer step sizes of 1 - 128. The fourth and fifth columns show the error between the coded model and

the original image at the various steps. The third column shows the squared quantization noise that would be expected if the original image were quantized directly at that step size and the values falling into each quantizer bucket were uniformly distributed. What is interesting is that for this experiment, the error produced by quantizing the sentinel points is comparable or less than the error expected from quantizing the image directly. This verifies that the algorithm for choosing sentinel points is operating fairly well for domains having shapes similar to those of Syn15. Figure 2.8 shows a reconstruction of Syn15 with the values of the sentinel points quantized using a *Qstep* of 64. The error introduced by quantization is perceptible but not objectionable even at this quantization level.

Q Step Size	Bits/Coef.	Reference Qnoise	MSE	PSNR
-	$\infty$	-	0.228	54.6
1	8	.25	0.417	51.9
2	7	.5	0.662	49.9
4	6	1.5	1.92	45.3
8	5	5.5	5.68	40.6
16	4	21.5	17.5	35.7
32	3	85.5	69.7	29.7
64	2	341.5	251.2	24.1
128	1	1365.5	937.8	18.4

Table 2.4  
Syn15 Implicit Point Quantization

The number of bits needed to code the sentinel point values can be further reduced by using prediction. Table 2.5 shows the result of using both quantization and prediction on the sentinel points of Syn16. The predictor used is the average value of the sentinel points received thus far for each domain. Since the first point received for each domain is not predicted, only the second and subsequent sentinel points benefit from prediction.

The most interesting feature of Table 2.5 is the interaction of actual image noise with quantization noise. The MSE of the reconstruction is not appreciably effected by quantization noise until its amplitude approaches the amplitude of the noise in the source image. For real images the analog to the introduced noise of Syn16 are features such as texture that are not piecewise smooth.

The significance of this is that pixel intensity functions can be quantized more heavily when the distortion before application of quantization is already high, such as when modeling a highly active image with only a few domain. Quantization distortion only becomes noticeable when its level approaches the error of the full precision piecewise approximation.

Quantizer Step Size	Code Bits	Entropy	MSE	PSNR
-	-	-	238.7	24.35
1	453	5.37	238.8	24.35
2	415	4.98	239.1	24.35
4	369	4.47	239.9	24.33
8	324	3.97	243.9	24.25
16	257	3.15	254.6	24.07
32	189	2.3	306.1	23.27
64	122	1.47	455.4	21.55
128	85	1.09	1070	17.84

Table 2.5  
Syn16 Implicit Point Quantization and Prediction

Perhaps the most significant assertion of Chapter 6 is that the smaller the domain size, the more its pixel intensity function can be quantized before introducing additional perceptible distortion. One reason for this is the nature of the extraction algorithm. Since smaller domains have a larger periphery to area ratio, they are preferentially merged to minimize the boundary length term of the model cost function. Clearly, the smaller the domain, the more its intensity must differ from its neighbors for it to persist during domain growing. Since small domains differ significantly in intensity from their neighbors, they can be quantized more heavily before distortion becomes noticeable. Experiments in human perception<sup>4</sup> have shown that the human eye has reduced sensitivity to intensity changes at low and high spatial frequencies. Since the smaller the domain the higher its associated spatial frequency, the more it can be quantized before introducing noticeable distortion.

The domain size and model fidelity dependencies combine to allow for significant quantization over a wide range of compression ratios. When compression is high, the overall error is already significant before quantization and therefore quantization can be quite heavy before it introduces additional distortion. When compression is low, most of the domains are quite

small and can be quantized more heavily. The amount of quantization applied to a domain of a given size is determined implicitly by a simple *size rule* known to both the encoder and decoder. No additional information is needed to specify the amount of quantization applied to a given domain.

### 2.3 Coding Experiments

In Chapter 8 we apply piecewise smooth coding to four commonly available images: Lena, Cameraman, Baboon, and Miss America. The experimental results are compared to the JPEG lossy algorithm for bit rates of approximately 0.125 to 1.0 bits/pixel.

Image	Piecewise-Smooth Bits	MSE	PSNR	JPEG Bits	MSE	PSNR
Syn15	1661	0.417	51.9	4088	1011	18.09
Syn16	1465	254.6	24.07	4104	1242	17.19

Table 2.6  
Coding Results for Syn15 and Syn16

To motivate these results, Table 2.6 gives coding results for Syn15 and Syn16. Unfortunately, the amount of fixed overhead in JPEG precludes it from coding at comparable rates. Of course, excellent piecewise smooth performance is expected from canonical examples.

Since JPEG is not designed to code below about 0.5 bits/pixel and changes in its design can increase its performance in the low bit rate regime, it is not fair to criticize its performance at low bit rates. What the data show, we believe, is that piecewise smooth coding is robust across a wider range of images than JPEG. What the data cannot show is the nature of the distortion produced by each method. We believe that the distortion introduced by piecewise smooth coding is significantly less objectionable than the block DCT artifacts of JPEG.

### 3. Moments of Two Dimensional Domains

Recall the first canonical example from Chapter 0 where we synthesized the image of Figure 2.1 from the polynomial intensity functions of Table 2.1. We now develop some powerful tools for addressing the inverse problem of recovering Table 2.1 from Figure 2.1. These tools are moment operators that manipulate sub-domains of an encompassing global domain. The moment operators are a general solution to the problem of fitting a piecewise-smooth surface through multidimensional data. Because this is a general problem we won't mention images in this chapter. We do remember that we are addressing the image problem, however, and all specific results are confined to the two-dimensional problem.

#### 3.1 A Digression

I still remember the summer before my high school junior year and the purchase of my first scientific calculator. I pored over every feature of that Texas Instruments SR51 and read the manual from cover to cover. I remember in particular its ability to turn random pairs of x and y coordinates into the slope and intercept of the best fit straight line through them.

One proceeded by entering in turn each coordinate pair into its respective register and pressing the  $\Sigma+$  key. A mistake or change of mind was rectifiable without doing everything over again by entering the data point to be removed and pressing  $\Sigma-$ . When all points had been entered the slope and intercept were a touch of the SLOPE and INTCP keys away.

My only significant use of this feature was to extract organic reaction rates from experimental data as a college sophomore. The venerable SR51 died shortly thereafter and although the HP41C bought as a replacement also had the same capability, I never had use for it again. Until recently.

#### 3.2 Moments

Linear regression against a single dimensional function is the simplest use of the method of *least squares*. The least squares metric simply minimizes the sum of the squares of the differences between the actual ordinates and their corresponding straight line predictions. It turns out that

the slope and intercept of the straight line that best approximates a set of data points in the least squares sense can be calculated by solving the following system of linear equations:<sup>5</sup>

$$\begin{bmatrix} \sum 1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} b \\ m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix},$$

where the  $x_i$  and  $y_i$  are the independent and dependent coordinates of the data points and the sums are over all the data points.

We develop this further later, but for now the interesting thing is that once we have performed the sums, we can calculate with a fixed number of operations the parameters of the best fit straight line no matter how many data points we have. We can also add and subtract single or groups of data points, not wasting any previously expended effort, and recalculate new approximation parameters with the same fixed number of operations.

The sums at the heart of this technique are commonly called *moments*. They are the discrete analogs of parameters such as center of mass and center of inertia that are commonly encountered in introductory calculus. As we extend the method to more than one dimension and to second and third order approximation functions, we encounter moments that become increasingly numerous and complex, so let's develop a shorthand sum notation.

We focus on the two dimensional problem and use  $x$  and  $y$  as the independent variables, with  $z$  as the dependent variable. The shorthand we use simply replaces the summation with the single letter  $S$  and makes the various powers of  $x$ ,  $y$  and  $z$  into subscripts. For example

$$\sum x_i^2 y_i^2 z_i^2 \xrightarrow{\text{Shorthand}} S_{x^2 y^2 z^2}.$$

Each problem order has two sets of moments. Since we weren't able to find terms for these sets in the literature, we coin new terms. The set of *natural moments* appears in the *natural moment matrix* on the left side of the equals sign and contains only sums of powers of the independent variables. The set of *forcing moments* comprises the *forcing moment matrix* on the right side and also

contain powers of the dependent variable. The natural moments are designated  $N$  and the forcing moments  $F$ . The union of  $N$  and  $F$  and  $S_{z^2}$  is  $M$  and is called the *moment set* of problem instance.  $S_{z^2}$  is not needed to find the least squares solution but is used later when determining the error between a domain's data and its polynomial approximation function.

### 3.3 Domain Operations

Our ultimate intent is not to find a single polynomial approximating function for a set of data, but to find a piecewise smooth approximation. To do this, we break the problem into a set of disjoint domains, the union of which is the global problem domain. We next define four operators that enable us to manipulate these domains via their moment sets:

- $add(M, p) \rightarrow \forall S \in M: S_{x^i y^j z^k} = S_{x^i y^j z^k} + x_p^i y_p^j z_p^k$
- $remove(M, p) \rightarrow \forall S \in M: S_{x^i y^j z^k} = S_{x^i y^j z^k} - x_p^i y_p^j z_p^k$
- $merge(M', M'') \rightarrow \forall S \in M': S'_i = S'_i + S''_i$
- $excise(M', M'') \rightarrow \forall S \in M': S'_i = S'_i - S''_i$

These operations let us efficiently add and subtract data points from a moment set and merge and separate two moment sets. Obviously, each of these operations is  $O(|M|)$  where  $|M|$  is the number of moment sums in a moment set. We see in the following sections that  $|M|$  for two dimensional domains is 10 for linear, 22 for quadratic and 38 for cubic approximating polynomials.

This chapter focuses only on fast methods for manipulating moment sets. The methods developed are used in later chapters. The greedy domain extraction algorithm of Chapter 3 uses the add and merge operations. The smoothing algorithm of Chapter 4 uses the add, excise and merge operations.

### 3.4 The Two Dimensional Linear System

In Section 3.2 the least squares system of equations for the one dimensional linear regression problem was given without justification. In two dimensions the problem becomes slightly more complex, but is still small enough to go through all the details that we gloss over for



higher order systems. Given a data generation function,  $z = p(x, y)$  over a domain  $D$  our task is to minimize point by point the square of the error between a two dimensional linear approximation function,  $f_1(x, y) = ix + jy + k$ , and the data generation function. In other words, we must find the coefficients  $i$ ,  $j$ , and  $k$  that minimize the following sum:

$$E = \sum_D (z_l - ix_l - jy_l - k)^2.$$

A local minimum for  $E$  exists where  $\frac{\partial E}{\partial i} = \frac{\partial E}{\partial j} = \frac{\partial E}{\partial k} = 0$ . Differentiating with respect to  $i$  yields:

$$0 = \sum_D 2(z_l - ix_l - jy_l - k)x_l.$$

Separating the sums and rearranging gives:

$$\sum_D x_l z_l = \sum_D ix_l^2 + \sum_D jx_l y_l + \sum_D kx_l.$$

Differentiating with respect to all three unknowns and using our sum shorthand notation gives the two dimensional linear least squares system:

$$\begin{bmatrix} S & S_y & S_x \\ S_y & S_{y^2} & S_{xy} \\ S_x & S_{xy} & S_{x^2} \end{bmatrix} \begin{bmatrix} k \\ j \\ i \end{bmatrix} = \begin{bmatrix} S_z \\ S_{ya} \\ S_{xz} \end{bmatrix}.$$

### 3.5 Second and Third Order Systems

The second order approximation function:

$$f_2(x, y) = fx^2 + gxy + hy^2 + ix + jy + k$$

yields the system:

$$\begin{bmatrix} S & S_y & S_x & S_{y^2} & S_{xy} & S_{x^2} \\ S_y & S_{y^2} & S_{xy} & S_{y^3} & S_{xy^2} & S_{x^2y} \\ S_x & S_{xy} & S_{x^2} & S_{xy^2} & S_{x^2y} & S_{x^3} \\ S_{y^2} & S_{y^3} & S_{xy^2} & S_{y^4} & S_{xy^3} & S_{x^2y^2} \\ S_{xy} & S_{xy^2} & S_{x^2y} & S_{xy^3} & S_{x^2y^2} & S_{x^3y} \\ S_{x^2} & S_{x^2y} & S_{x^3} & S_{x^2y^2} & S_{x^3y} & S_{x^4} \end{bmatrix} \begin{bmatrix} k \\ j \\ i \\ h \\ g \\ f \end{bmatrix} = \begin{bmatrix} S_z \\ S_{yz} \\ S_{xz} \\ S_{y^2z} \\ S_{xyz} \\ S_{x^2z} \end{bmatrix}$$

The third order approximation function:

$$f_3(x, y) = ax^3 + bx^2y + cxy^2 + dy^3 + fx^2 + gxy + hy^2 + ix + jy + k$$

yields the system:

$$\begin{bmatrix} S & S_y & S_x & S_{y^2} & S_{xy} & S_{x^2} & S_{y^3} & S_{xy^2} & S_{x^2y} & S_{x^3} \\ S_y & S_{y^2} & S_{xy} & S_{y^3} & S_{xy^2} & S_{x^2y} & S_{y^4} & S_{xy^3} & S_{x^2y^2} & S_{x^3y} \\ S_x & S_{xy} & S_{x^2} & S_{xy^2} & S_{x^2y} & S_{x^3} & S_{xy^3} & S_{x^2y^2} & S_{x^3y} & S_{x^4} \\ S_{y^2} & S_{y^3} & S_{xy^2} & S_{y^4} & S_{xy^3} & S_{x^2y^2} & S_{y^5} & S_{xy^4} & S_{x^2y^3} & S_{x^3y^2} \\ S_{xy} & S_{xy^2} & S_{x^2y} & S_{xy^3} & S_{x^2y^2} & S_{x^3y} & S_{xy^4} & S_{x^2y^3} & S_{x^3y^2} & S_{x^4y} \\ S_{x^2} & S_{x^2y} & S_{x^3} & S_{x^2y^2} & S_{x^3y} & S_{x^4} & S_{x^2y^3} & S_{x^3y^2} & S_{x^4y} & S_{x^5} \\ S_{y^3} & S_{y^4} & S_{xy^3} & S_{y^5} & S_{xy^4} & S_{x^2y^3} & S_{y^6} & S_{xy^5} & S_{x^2y^4} & S_{x^3y^3} \\ S_{xy^2} & S_{xy^3} & S_{x^2y^2} & S_{xy^4} & S_{x^2y^3} & S_{x^3y^2} & S_{xy^5} & S_{x^2y^4} & S_{x^3y^3} & S_{x^4y^2} \\ S_{x^2y} & S_{x^2y^2} & S_{x^3y} & S_{x^2y^3} & S_{x^3y^2} & S_{x^4y} & S_{x^2y^4} & S_{x^3y^3} & S_{x^4y^2} & S_{x^5y} \\ S_{x^3} & S_{x^3y} & S_{x^4} & S_{x^3y^2} & S_{x^4y} & S_{x^5} & S_{x^3y^3} & S_{x^4y^2} & S_{x^5y} & S_{x^6} \end{bmatrix} \begin{bmatrix} k \\ j \\ i \\ h \\ g \\ f \\ d \\ c \\ b \\ a \end{bmatrix} = \begin{bmatrix} S_z \\ S_{yz} \\ S_{xz} \\ S_{y^2z} \\ S_{xyz} \\ S_{x^2z} \\ S_{y^3z} \\ S_{xy^2z} \\ S_{x^2yz} \\ S_{x^3z} \end{bmatrix}$$

Note the way the linear system is embedded in the quadratic system which is embedded in the cubic system. We exploit this arrangement to solve singular systems in the next section.

### 3.6 Solving Least Squares Systems

#### 3.6.1 Closed Form Solution for the Linear Problem

The two dimensional linear system is small enough so that a closed form solution is tractable.

If the determinant of the  $S$  (left hand) matrix is:

$$D = S \cdot S_{y^2} \cdot S_{x^2} - S \cdot S_{xy}^2 - S_y^2 \cdot S_{x^2} + 2 \cdot S_y \cdot S_x \cdot S_{xy} - S_x^2 \cdot S_{y^2}$$

then the solution is:

$$\begin{bmatrix} k \\ j \\ i \end{bmatrix} = \frac{1}{D} \begin{bmatrix} S_z \cdot (S_{y^2} \cdot S_{x^2} - S_{xy}^2) + S_{yz} \cdot (S_x \cdot S_{xy} - S_y \cdot S_{x^2}) + S_{xz} \cdot (S_y \cdot S_{xy} - S_x \cdot S_{y^2}) \\ S_z \cdot (S_x \cdot S_{xy} - S_y \cdot S_{x^2}) + S_{yz} \cdot (S \cdot S_{x^2} - S_x^2) + S_{xz} \cdot (S_x \cdot S_y - S \cdot S_{xy}) \\ S_z \cdot (S_y \cdot S_{xy} - S_x \cdot S_{y^2}) + S_{yz} \cdot (S_x \cdot S_y - S \cdot S_{xy}) + S_{xz} \cdot (S \cdot S_{y^2} - S_y^2) \end{bmatrix}$$

This solution requires 39 multiplications, 10 additions and 1 division. Difficulties are presented by  $\frac{1}{D}$  term found in the solution for all three coefficients. Some domains, a trivial example being a domain with a single data point, generate singular first order  $S$  matrices and the determinant is zero. Domains whose  $S$  matrix is singular do not contain enough data points to uniquely determine all three polynomial coefficients.

For example a domain whose data points all have identical  $x$  coordinates would have difficulty generating a coefficient for the  $y$  term in the approximating polynomial. What is desired for such domains is for the solution procedure to deliver coefficients for which it has information and zero for the others. In the pathological example of the single point domain, only  $k$  would have a non-zero value: the value of the dependent variable at that point. The next section develops a solution method with the desired characteristics.

### 3.6.2 Cholesky Factorization of Symmetric Positive Semidefinite Systems

#### 3.6.2.1 Symmetric Positive Definite Systems

The Cholesky factorization for symmetric positive definite matrices is a variation of  $\mathbf{LU}$  factorization that takes advantage of symmetry to reduce the operation count by a factor of two, since  $\mathbf{U} = \mathbf{L}^T$ . The method uses  $O(\frac{n^3}{6})$  multiplications and additions.

Since the matrix is positive, square roots can be used to calculate the diagonal elements of  $\mathbf{L}$ . The square roots can be avoided by a variation of the technique that leads to a factorization of

the form  $\mathbf{LDL}^T$  where  $\mathbf{L}$  is a lower triangular matrix with unit diagonal elements and  $\mathbf{D}$  is a diagonal matrix

The systems of Sections 3.4 and 3.5 are called the *normal* equations for a least squares problem. The matrices of normal equations are known to be positive but are not necessarily definite. As described previously, they can be singular which results in zero pivots. Zero pivots lead to problems for factorization schemes that are usually solved by full or partial pivoting. A permutation is applied to the system before each row reduction that hopefully can bring a non-zero element to the pivot position.

Pivoting is only successful in solving matrices that are non-singular. Pushing a near zero pivot further down into the matrix only delays things unless the matrix is non-singular. For singular matrices a zero pivot is eventually encountered that a permutation cannot remove. The *singular value decomposition*<sup>6</sup> is a method for dealing with singular matrices.

### 3.6.2.2 The Singular Value Decomposition

The singular value decomposition works on any matrix but, we limit the discussion to its use with symmetric positive semidefinite matrices. For such a matrix,  $\mathbf{S}$ , a singular value decomposition factors it into the form  $\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$  where  $\mathbf{Q}$  has as its columns the eigenvectors of  $\mathbf{S}$  and  $\mathbf{\Lambda}$  is a diagonal matrix with the eigenvalues of  $\mathbf{S}$  along its diagonal.  $\mathbf{Q}$  is also orthonormal,  $\mathbf{Q}^T\mathbf{Q} = \mathbf{I}$ , so solution of the least squares system,  $\mathbf{S}\mathbf{x} = \mathbf{b}$  after decomposition is  $\mathbf{x} = \mathbf{Q}\mathbf{\Lambda}^{-1}\mathbf{Q}^T\mathbf{b}$ .

Of course since  $\mathbf{\Lambda}$  can have zero elements along the diagonal and therefore be singular,  $\mathbf{\Lambda}^{-1}$  can fail to exist. Neglecting zero diagonal elements of  $\mathbf{\Lambda}$ ,  $\mathbf{\Lambda}^{-1}$  is simply the matrix formed by replacing each diagonal element of  $\mathbf{\Lambda}$  with its multiplicative inverse. The trick for handling zero or near zero diagonal elements of  $\mathbf{\Lambda}$  is to just replace them with zero in  $\mathbf{\Lambda}^{-1}$ . The solution resulting from this replacement is provably of minimum length<sup>5</sup>.

Of the many polynomials that can approximate the underdetermined system, we want the one with the smallest coefficients, and the minimum length supplies just that. Also, zeroed coefficients of the minimum length solution are the ones for which the system contains the least information, exactly what we want.

### 3.6.2.3 Cholesky Factorization Extended to Semidefinite Systems

The singular value decomposition is significantly more complex than the Cholesky factorization and even though in most cases its asymptotic complexity is equivalent, its overhead makes it significantly slower for small matrices such as ours. For this reason, we develop a procedure to extend the Cholesky factorization to positive semidefinite systems by applying the singularity removal technique of the singular value decomposition.

The Cholesky method starts at the upper left of a matrix and proceeds down the diagonal with a progressively wider wavefront. The procedure is done *in place* with the entries below the diagonal replaced with **L**, the diagonal entries replaced with **D**, and the entries above the diagonal replaced with **DL<sup>T</sup>**.

$$\text{For } i < j, s_{ij} = s_{ij} - \sum_{k < i} s_{ik} \cdot s_{kj}.$$

$$\text{For } i > j, s_{ij} = \frac{s_{ji}}{s_{jj}}.$$

$$\text{For } i = j, s_{ii} = s_{ii} - \sum_{j < i} s_{ij} \cdot s_{ji}.$$

Once the **S** matrix has been factored, the system **Sx = b** is solved by forward (*i* increasing),

$$b_i = b_i - \sum_{j < i} s_{ij} b_j$$

and then backward (*i* decreasing),

$$b_i = \frac{b_i - \sum_{j>i} s_{ij} b_j}{s_{ii}}$$

substitution, with *in place replacement* of  $\mathbf{b}$ .

Since  $\mathbf{S}$  is positive semidefinite, we know that both its eigenvalues and pivots are either positive or zero. Additionally, if an eigenvalue is zero, its corresponding pivot is zero. Since the Cholesky method delivers the pivots of  $\mathbf{S}$  into the diagonal of the solution, the  $s_{ii}$  are constrained to be greater than or equal to zero. Our first modification of the Cholesky method is to *replace a diagonal element,  $s_{ii}$ , with zero if its factored value is less than a stability parameter  $\alpha$* :

$$s_{ii} < \alpha \Rightarrow s_{ii} = 0.$$

Further, just as in the singular value decomposition, we replace anything divided by a zero  $s_{ii}$  with zero. Now for some justification.

In  $\mathbf{S}$ , each column is contributed by a particular polynomial coefficient and each row is contributed by differentiation with respect to a particular coefficient. If the domain does not have enough points to uniquely define a coefficient, we would like for the system to behave as if we had not included its polynomial term in the first place. Ideally, the system would reduce to the principle minor of the singular value and become positive definite.

Let's first look at  $\mathbf{S}$ . In Section 3.5 we defined the rows and columns  $\mathbf{S}$  in a particular manner so that lower order systems would always be embedded in those of higher order. Since the Cholesky method starts at the upper left and works down and to the right, if there is a singular coefficient, it is always of the highest possible order. For example, if all points of a domain have only two different  $y$  coordinates, it is not possible to uniquely determine coefficients for  $y^0$ ,  $y^1$ , and  $y^2$ . By the arrangement of the Cholesky system, we assure that the singular value arises in the  $y^2$  column of the Cholesky matrix.

Next, we show that a singular value does not influence the values of any entries in  $\mathbf{S}$  that are not on the row or column of the singular value. Figure 3.1 shows an example quadratic system whose  $y^2$  term is singular or nearly so. When solving the system, the diagonal element corresponding to  $y^2$ , labeled  $0$ , is less than  $\alpha$  and is set to zero. The elements  $0'$  are forced to zero since all elements below the diagonal are divided by the diagonal element of their column. The elements  $0''$  could be set to zero by reflecting  $0'$  across the diagonal but it is not necessary to do this directly. Any use of an above the diagonal element in the equations for above the diagonal or on the diagonal elements is multiplied with its transpose image across the diagonal. Since the  $0''$  elements are the reflections of the  $0'$  elements, the  $0''$  elements are *virtual zeros*.

$$\begin{bmatrix} S & S_y & S_x & S_{y^2} & S_{xy} & S_{x^2} \\ S_y & S_{y^2} & S_{xy} & S_{y^3} & S_{xy^2} & S_{x^2y} \\ S_x & S_{xy} & S_{x^2} & S_{xy^2} & S_{x^2y} & S_{x^3} \\ S_{y^2} & S_{y^3} & S_{xy^2} & 0 & 0'' & 0'' \\ S_{xy} & S_{xy^2} & S_{x^2y} & 0' & S_{x^2y^2} & S_{x^3y} \\ S_{x^2} & S_{x^2y} & S_{x^3} & 0' & S_{x^3y} & S_{x^4} \end{bmatrix}$$

Figure 3.1  
Zero Propagation

When forward substituting, the mixture elements  $s_{ij}$  are those elements of  $\mathbf{S}$  below the diagonal since  $i > j$ . These are the  $0'$  elements. When backward substituting, the mixture elements are the  $0''$ . Once again these are virtual zeros, because the only use of these elements is in the calculation of the coefficient of  $y^2$ . But this coefficient is divided by its diagonal element, resulting in zero. Since the coefficient of  $y^2$  is zero and solution for the other coefficients are not effected by the presence of the  $y^2$  entries in the system, it is as if they do not exist. The solution is the same as if we had not initially included the  $y^2$  moments in the system!

### 3.6.2.4 Examples

The significance of the previous result is difficult to appreciate without some examples. Figure 3.2 shows four sample domains. Domains **b** and **c** are second order singular and **a**, and **d** are not. In the following experiments the *forcing function*

$$z = 100 + y - x - y^2 - 2xy + x^2$$

is placed over each domain. The name forcing function indicates that it is the source of the forcing moments. It is simply the data generation function for our example domains. Since the coordinate of the upper left corner point of each domain is (0,0), a value of 100 for the constant term of the forcing function assures that it remains positive over all four of our sample domains.

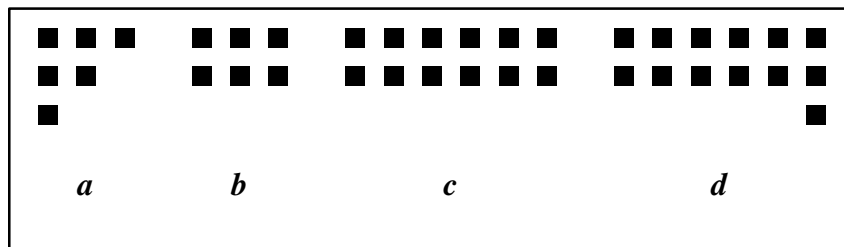


Figure 3.2  
Sample Domains

The second order two-dimensional normal equations,  $\mathbf{Sx} = \mathbf{b}$ , for the given forcing function over domain **a** are:

$$\begin{bmatrix} 6 & 4 & 4 & 6 & 1 & 6 \\ 4 & 6 & 1 & 10 & 1 & 1 \\ 4 & 1 & 6 & 1 & 1 & 10 \\ 6 & 10 & 1 & 18 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 6 & 1 & 10 & 1 & 1 & 18 \end{bmatrix} \begin{bmatrix} k \\ j \\ i \\ h \\ g \\ f \end{bmatrix} = \begin{bmatrix} 598 \\ 394 \\ 402 \\ 590 \\ 98 \\ 606 \end{bmatrix}.$$

Note that the natural moments populate **S** and the forcing moments populate **b**. The Cholesky factorization and solution for domain **a** are:



$$\mathbf{S} = \begin{bmatrix} 6 & 4 & 4 & 6 & 1 & 6 \\ .67 & 3.33 & -1.67 & 6 & .33 & -3 \\ .67 & -5 & 2.5 & 0 & .5 & 4.5 \\ 1 & 1.8 & 0 & 1.2 & -.6 & .4 \\ .17 & .1 & .2 & -.5 & .4 & -.4 \\ 1 & -9 & 1.8 & .33 & -1 & .67 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 100 \\ 1 \\ -1 \\ -1 \\ -2 \\ 1 \end{bmatrix}.$$

Note the correct recovery of the coefficients of the forcing function in  $\mathbf{b}$ . Domain  $\mathbf{a}$  is the smallest possible second order non-singular domain. The solution to domain  $\mathbf{b}$ , also having six elements is:

$$\mathbf{S} = \begin{bmatrix} 12 & 6 & 30 & 6 & 15 & 110 \\ .5 & 3 & 0 & 3 & 7.5 & 0 \\ 2.5 & 0 & 35 & 0 & 17.5 & 175 \\ .5 & 1 & 0 & 0 & 0 & 0 \\ 1.25 & 2.5 & .5 & 0 & 8.75 & 0 \\ 9.17 & 0 & 5 & 0 & 0 & 74.67 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 100 \\ 0 \\ -1 \\ 0 \\ -2 \\ 1 \end{bmatrix}.$$

Note the singularity developing in the  $y^2$  column of  $\mathbf{S}$ . This is due to  $\mathbf{b}$  having insufficient extent in the  $y$  direction to uniquely determine all three  $y$  coefficients.

The solution for domain  $\mathbf{c}$ :

$$\mathbf{S} = \begin{bmatrix} 24 & 12 & 132 & 12 & 66 & 1012 \\ .5 & 6 & 0 & 6 & 33 & 0 \\ 5.5 & 0 & 286 & 0 & 143 & 3146 \\ .5 & 1 & 0 & 0 & 0 & 0 \\ 2.75 & 5.5 & .5 & 0 & 71.5 & 0 \\ 42.17 & 0 & 11 & 0 & 0 & 2669.33 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 100 \\ 0 \\ -1 \\ 0 \\ -2 \\ 1 \end{bmatrix}$$

shows a similar result for a larger domain of the same shape. For both of these examples, the  $y$  as well as the  $y^2$  coefficient turned out to be zero. This is due to the forcing function over the domain, not the geometry of the domain itself.

Domain  $\mathbf{d}$  adds another point to  $\mathbf{c}$  and removes the singularity:

$$\mathbf{S} = \begin{bmatrix} 25 & 14 & 143 & 16 & 88 & 1133 \\ .56 & 8.16 & 7.92 & 11.04 & 60.72 & 113.52 \\ 5.72 & .97 & 307.35 & 7.76 & 185.71 & 3452.52 \\ .64 & 1.35 & .03 & 2.63 & 10.84 & 24.08 \\ 3.52 & 7.44 & .6 & 4.13 & 71.5 & 0 \\ 45.32 & 13.91 & 11.23 & 9.17 & 0 & 2669.33 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 100 \\ 1 \\ -1 \\ -1 \\ -2 \\ 1 \end{bmatrix}.$$

All examples thus far have been solved with a stability parameter,  $\alpha$ , of  $\frac{S}{32}$ , where  $S$  is the

upper left element of  $\mathbf{S}$ . If we reduce  $\alpha$  to  $\frac{S}{8}$ , the solution for domain  $\mathbf{d}$  is:

$$\mathbf{S} = \begin{bmatrix} 25 & 14 & 143 & 16 & 88 & 1133 \\ .56 & 8.16 & 7.92 & 11.04 & 60.72 & 113.52 \\ 5.72 & .97 & 307.35 & 7.76 & 185.71 & 3452.06 \\ .64 & 1.35 & .03 & 0 & 10.84 & 24.08 \\ 3.52 & 7.44 & .6 & 0 & 116.21 & 99.35 \\ 45.32 & 13.91 & 11.23 & 0 & .85 & 2805.17 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 99.8 \\ .3 \\ -.91 \\ 0 \\ -2.09 \\ .99 \end{bmatrix}.$$

As can be seen, the  $y^2$  term of the polynomial has been forced to zero by the singular value correction procedure. The solution is just the best least squares coefficients for the five remaining polynomial terms. What is especially interesting is how the  $y^2$  term has the smallest pivot and is the first to be eliminated by raising the value of  $\alpha$ . This is exactly the intuitive result that one would expect from domain  $\mathbf{d}$ . Looking again at the  $\alpha = \frac{S}{32}$  solution to  $\mathbf{d}$ , we can see that the order in which coefficients are zeroed as the stability factor increases is:

$y^2 \rightarrow y \rightarrow xy \rightarrow x \rightarrow x^2$ . The constant term cannot be eliminated as its diagonal element of  $\mathbf{S}$  remains unchanged by the procedure

### 3.7 Error Function

In addition to solving for the least squares polynomial approximation for a domain, we need to solve for the error between the approximating polynomial and the actual data values. The sum of the squares of the differences between the data points and the approximating polynomial is:

$$E_1 = \sum_D (z_l - ix_l - jy_l - k)^2 .$$

Expanding the sum and using our shorthand sum notation, the total squared error for the two dimensional linear approximation is:

$$E_1 = S_{z^2} + k^2 S + 2jkS_y + j^2 S_{y^2} + 2ikS_x + 2ijS_{xy} + i^2 S_{x^2} - 2(kS_z + jS_{yz} + iS_{xz}) .$$

The error for the quadratic model is:

$$\begin{aligned} E_2 = & S_{z^2} + \\ & k^2 S + 2jkS_y + (2hk + j^2)S_{y^2} + 2hjS_{y^3} + h^2 S_{y^4} + 2ikS_x + 2(ij + gk)S_{xy} + \\ & 2(hi + gj)S_{xy^2} + 2ghS_{xy^3} + (2fk + i^2)S_{x^2} + 2(fj + gi)S_{x^2y} + (2fh + g^2)S_{x^2y^2} + \\ & 2fS_{x^3} + 2fgS_{x^3y} + f^2 S_{x^4} \\ & - 2(kS_z + jS_{yz} + iS_{xz} + hS_{y^2z} + gS_{xyz} + fS_{x^2z}) \end{aligned}$$

and the error for the cubic model is:

$$\begin{aligned}
E_3 = & S_{z^2} + \\
& k^2 S + 2kjS_y + (j^2 + 2kh)S_{y^2} + 2(kd + jh)S_{y^3} + (2jd + h^2)S_{y^4} + 2hdS_{y^5} + d^2S_{y^6} + \\
& 2kiS_x + 2(ji + kg)S_{xy} + 2(jg + hi + kc)S_{xy^2} + 2(jc + hg + di)S_{xy^3} + 2(hc + dg)S_{xy^4} + \\
& 2dcS_{xy^5} + (2kf + ii)S_{x^2} + 2(kb + jf + ig)S_{x^2y} + (2(jb + hf + ic) + g^2)S_{x^2y^2} + \\
& 2(df + gc + hb)S_{x^2y^3} + (c^2 + 2db)S_{x^2y^4} + 2(ka + if)S_{x^3} + 2(gf + ja + ib)S_{x^3y} + \\
& 2(gb + ha + cf)S_{x^3y^2} + 2(da + cb)S_{x^3y^3} + (2ia + f^2)S_{x^4} + 2(ga + fb)S_{x^4y} + \\
& (b^2 + 2ca)S_{x^4y^2} + 2faS_{x^5} + 2baS_{x^5y} + a^2S_{x^6} - \\
& (2kS_z + jS_{yz} + iS_{xz} + hS_{y^2z} + gS_{xyz} + fS_{x^2z} + dS_{y^3z} + cS_{xy^2z} + bS_{x^2yz} + aS_{x^3z})
\end{aligned}$$

### 3.8 Space/Precision Issues

Since the domain management methods developed here are designed for use in extracting a piecewise smooth approximation of a set of data, all domains have a common coordinate reference. For this reason, domains that are far from the origin can have quite large moments even if the domain itself is quite small. For example, in the third order system the natural moments contain powers of  $x$  and  $y$  up to six. For a single point domain at 100,100 its largest natural moments have a magnitude of  $100^6$ .

For effective use of cubic systems, 64 bit integers must be used to keep the moment sums. The maximum domain coordinates used should be less than 1000. With 39 moments per domain, 312 bytes of memory must be available for each domain in use. For good convergence of the Cholesky factorization, 80 bit floating point (64 bit mantissa) precision is necessary.

For second order systems, at least double precision floating numbers (48 bit mantissa) are necessary to maintain moment sums. Double precision calculations are necessary for reliable Cholesky convergence. The 23 second order moments need 184 bytes of memory per domain.

First order moments can generally be kept in 32 bit integers. Double precision should still be used for Cholesky convergence. The memory requirement for the 10 moments is 40 bytes per domain.

Thus far, the method has been tested on zero origin global domains up to 352x288 data points with ranges of 0 255. The range should be extendible to 32 bits before it becomes the limiting factor. Further increases in precision may be necessary for larger domains or ranges.

### **3.9 Summary**

We developed moment operators for use in finding least squares piecewise-polynomial approximations of multidimensional data. We introduced terminology for describing the components of multidimensional least squares normal equations. The natural moment matrix is comprised of moments of the independent variables. The forcing moment matrix is comprised of moments of both the independent and dependent variables. We developed complete natural moment matrices, forcing moment matrices and least squares error functions for first, second, and third order two-dimensional polynomials. We extended the Cholesky factorization of symmetric positive definite matrices to symmetric positive semidefinite matrices. We used the modified Cholesky factorization to determine the polynomial coefficients supported by a domain.

## 4. Domain Extraction

### 4.1 Motivation

For Piecewise Smooth Coding to be useful, an efficient method for extracting an optimal set of domains and pixel intensity functions from an original image must be available. If we refer to a proposed set of domains and functions as an image model, to determine optimality an extraction procedure must balance two different measures. First, how well does the model approximate the original image? Second, how much does it cost to code the model?

To determine how well a proposed model approximates the original image we must measure the error between the image rendered from the model and the original. Since any extraction procedure must accept or reject many different proposals in the process of obtaining a near optimal one, the error comparison operation must be quite fast. The methods of Chapter 3 are one way to achieve  $O(1)$  error comparison.

The final arbiter of a model's cost is the number of bits necessary to code it. Unfortunately, since a great number of cost measurements must be made by any extraction procedure, it is not tractable to completely code each proposed model to obtain its cost. However, the exact cost of a model is generally not needed to obtain an approximately optimal solution. If an approximate functional dependence of the total model cost on each domain and intensity function element is known, the function can be used in the extraction procedure.

An extraction procedure must balance the two competing requirements of model fidelity and model code cost. It certainly is not entirely obvious how this can be done, so it is useful to examine the literature for prior work.

### 4.2 Background

The closest area of study to domain extraction is image segmentation. Segmentation has been described extensively in the literature<sup>2,3</sup> and is similar to domain extraction in that an image is partitioned into regions or segments that align (somewhat) with objects in the image. A wide variety of methods have been documented. They generally fall into four categories: histogram

clustering, region growing, split and merge, and boundary finding and linking techniques. One drawback of most reported techniques is that algorithm parameters must be adjusted for each segmented image for reasonable results to be achieved. More recently, several schemes have been proposed that are based upon global optimization techniques and attempt to remove this parameter adjustment problem.

Perhaps the first reported use of global optimization techniques was by Gemen and Gemen<sup>7</sup> who used a Bayesian model and simulated annealing for image restoration. LeClerc<sup>8</sup> used Minimum Descriptive Length arguments to arrive at a similar model formulation and used continuation functions as his optimization strategy. While potentially highly parallel, these first descriptions are not directly applicable to segmentation of arbitrary images since the techniques partition an image into a set of predefined equivalence classes. Prior knowledge of potential equivalence classes is only available in certain carefully constructed problem instances.

Marques, Gasull, Reed and Kunt<sup>9</sup> show a boundary relaxation technique that is a descendent of the approach of Gemen and Gemen. While this strategy is useful for optimizing the boundaries of an existing partition, it cannot be easily extended to find the optimal number of regions in a segmentation. Sheinvald, Dom and Niblack<sup>10</sup> propose a greedy strategy for region growing. Minimum Descriptive Length arguments are used to derive a cost function that drives the region merging process.

Kwon and Chellappa<sup>11</sup> describe an interesting technique for region growing that uses separate methods to grow smooth and textured regions. The smooth regions that overlap at least 50% with a textured region are classified as textured. Smooth regions are approximated by second order polynomials and are grown by merging adjacent regions that cause an increase in approximation error below a certain threshold. This is a recent result and it shows how the use of global optimization techniques for image segmentation is still not widespread. Their experimental results testify to the difficulty of controlling the growth of appropriate second order regions using simple threshold merge operations.

Another method that deserves particular analysis is the variable order surface fitting algorithm of Besl and Jain<sup>12</sup>. It is the most elaborate of previously reported methods that model regions

with smooth intensity functions. The method uses local image characteristics to classify the intensity surface over each pixel. A subsequent step maps the local surfaces into larger surfaces while attempting to minimize the surface order. Because this method essentially takes local derivatives, its performance degrades rapidly in the presence of noise. It was developed to segment low-noise images obtained from a distance measuring apparatus.

### 4.3 Greedy Domain Growing

This Chapter embodies a greedy domain growing algorithm for image partitioning. It is similar to the method of Sheinvald<sup>10</sup> in that it uses an underlying image model that contains both bulk and boundary components. It is more general in the sense that the exact form of the image model is not predetermined. In keeping with the terminology of other chapters, we refer to a region as a *domain* and to a segmentation as a *partition*.

An initial partition is formed by placing each pixel into its own domain. Domains are *grown* by *merging* with adjacent domains. Each domain has an *identity*. The identity of a domain is simply the data structure assigned to it by the algorithm. After each merge the smaller of the two merged domains loses its identity.

The greedy order is determined by an associated *cost function*. The algorithm is independent of the exact form of this cost function. The identity of each domain holds *private* data that is managed by the cost function's *image model*. The *bulk component* of the image model is *consulted* via a *callback routine* when two domains are merged. A consultation allows the image model to update itself as the partition changes.

Additionally, the *boundary component* of the image model is consulted when traversing the partition boundaries. Each traversal yields a *transient* model of the boundary separating adjacent pairs of domains. The identities of two domains and the transient model of their boundary are made available as parameters to the cost function. The cost function determines the merge cost of two domains and thereby the overall merge order.



## 4.4 Implementation

### 4.4.1 Concepts

There are three key areas that must be addressed in any efficient implementation of a domain growing algorithm: membership, topology, and priority. If a domain is defined as a set, the members of the set are its pixels. Topology refers to connectedness. For instance, given a pixel, what are its neighbor pixels? Given a domain, what are its neighbor domains and pixels? Priority determines the order in which domains are merged.

A *set* contains a list of its members and each member knows its set. Each domain identity includes a set of its pixels. Since a domain is a set, a *pixel* can be a member of only one domain. Each pixel also knows its own location in a *field*. The field's function is to provide direct access to a pixel's near neighbors and is implemented as a two dimensional array.

Whereas pixel topology is easily embodied in the field concept, deduction of domain topology requires a more powerful technique. The *traveler* can traverse the outer boundary of a *home domain*. A boundary traversal enumerates all pixel pairs that are on either side of the imaginary boundary line that separates a domain from its neighbors.

An *accumulator* is a fixed size set with an *age tag* for each of its entries. An accumulator's *global age tag* replaces an entry's age tag when that entry is initially *added* or when a current entry is *updated*. If an accumulator's global age tag is incremented, entries updated or added subsequent to the increment are *younger* than those added or updated prior to the increment. Accumulator entries cannot be older than a *maximum age*. Entries older than their accumulator's maximum age are *invalid*. An accumulator has a *current size* that is less than its maximum. An accumulator is emptied simply by incrementing its maximum age and resetting its current size. An accumulator's contents can be *enumerated*. All accumulator operations except for enumeration are  $O(1)$ . Accumulator enumeration is  $O(N)$  where  $N$  is the current size.

Each domain identity contains an *accumulator location tag* that locates its accumulator entry if it has one. Because of this, a domain can have only one associated accumulator entry at a time. The accumulator tag makes *accumulator membership query* an  $O(1)$  operation. A traveler

*accumulates* neighboring domains (adds them to an accumulator) as it traverses the home domain's boundary. Each accumulator entry also contains a private data structure that holds the transient boundary model of its associated domain. As discussed previously this transient model is managed by consultation with the boundary component of the image model.

The traveler and associated accumulator concepts only apply to the outer boundary of a domain. A complete topological description of the neighborhood of a domain must also include interior neighbors or *holes*. Each domain identity also contains a set of its holes. Every domain is either in the global domain set or in another domain's hole set.

If north on the image plane is defined as pointing toward the upper part of the image, one of a domain's uppermost pixels is designated its *northernmost*. A domain may have more than one pixel of the same maximum latitude. Any one of these pixels can be designated northernmost. Since the northernmost pixel is used as an anchor point when traversing the boundary of the domain, the key requirement is that *no pixel of a domain may be further north than its northernmost*.

The complete topological description of a domain partition is quite large. A mechanism is needed for focusing attention on areas that contain adjacent domains that are similar via some cost measure. A *priority heap*<sup>13</sup> is used to hold one entry for each domain in the partition. Each heap entry contains the best neighboring merge candidate of its corresponding domain. The head of the heap therefore provides immediate access to the most promising pair of merge candidates.

Maintenance of the heap as domains are merged is non-trivial. To aid in this function, each domain identity contains a *heap tag* that can directly access its corresponding heap entry. After two domains are merged, the local boundary is traversed and the heap entries for effected neighbors are updated. This may necessitate the movement of the associated heap entries. The heap has special operations that facilitate the updating and possible movement of heap elements and the extraction of members not at the heap's head. In Section 4.4.2.1 we describe exactly how these special operations are used.

#### 4.4.2 Algorithms

The domain growing process proceeds in a greedy fashion, the order being determined by an associated cost function. Initial domains are formed by placing each pixel in its own domain. Optionally, a simpler seed growing strategy can be used before use of the greedy merge procedure. The priority heap is initialized with one entry for each domain. The following sections detail each operation of the algorithm. The algorithm is summarized in Figure 4.3 at the end of the detail descriptions.

##### 4.4.2.1 Heap Maintenance

A priority heap is used to maintain an ordering of possible merge candidates. In addition to the standard operations of insert, peek head, and extract head, the heap object contains two additional operations to update heap entries of neighboring domains when two domains are merged. A generalized *extract* method is used to remove the heap entry of the domain that is merged to the domain whose heap entry is at the head of the heap. The *adjust* method is used to move heap entries for those domains neighboring two merged domains whose best merge cost may have been effected by the merger. The update and generalized extract methods are both  $O(\log(N))$ .

Each domain identity of the partition contains a tag that is an index into the greedy merge heap for the entry associated with the domain. When a heap entry for a domain needs to be adjusted, the tag locates the entry.

The key operation performed by most heap methods is swapping of heap entries. The swap method knows the form of heap entries and updates their heap tags as their associated heap entries move. Heap entries are kept small to facilitate movement.

##### 4.4.2.2 Traveling

Figure 4.1 shows an example counterclockwise boundary traversal. The home domain's northernmost pixel anchors the boundary traversal. The boundary separator between the home domain's northernmost pixel and the pixel immediately to its north is a traversal's

starting point. For a counterclockwise traversal, the initial direction of travel is west, and east for a clockwise traversal.

The key traveling operation is **next\_move**. Each **next\_move** operation moves to the next separator location on the home domain's periphery. If pixels are thought of as squares, a separator is defined as an imaginary line that separates an edge of a pixel from the opposing edge of its neighbor. A single pixel can be surrounded by at most four separators, and at most three if the domain containing the pixel contains more than one pixel.

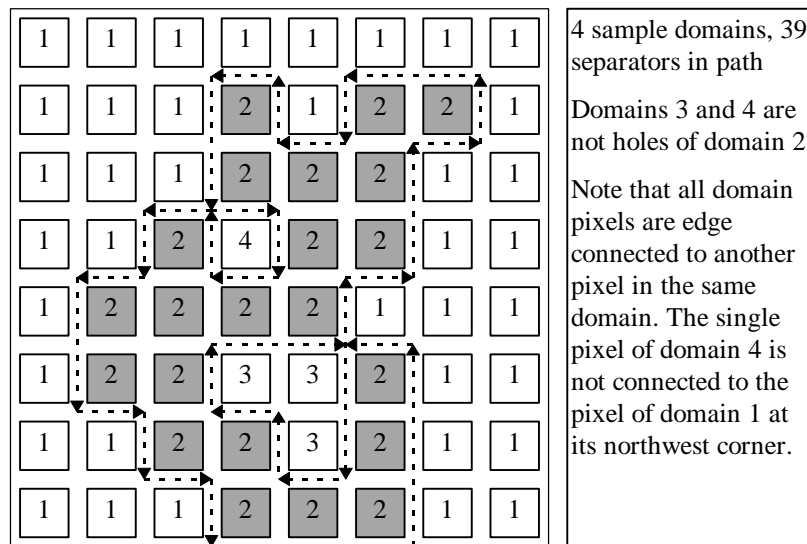


Figure 4.1  
 Traveling a Domain Boundary

The **next\_move** operation follows the boundary as one would follow a maze. When traveling counterclockwise an attempt to turn right is first made. If a right turn leaves the boundary, the straight ahead direction is checked. Finally, if proceeding straight ahead leaves the boundary, a left turn is made. A right turn is possible if both the pixel just ahead of the anchor pixel and the pixel just ahead and to the right are in the same domain as the anchor pixel. Proceeding straight ahead is possible only if the pixel just ahead is in the same domain as the anchor pixel.

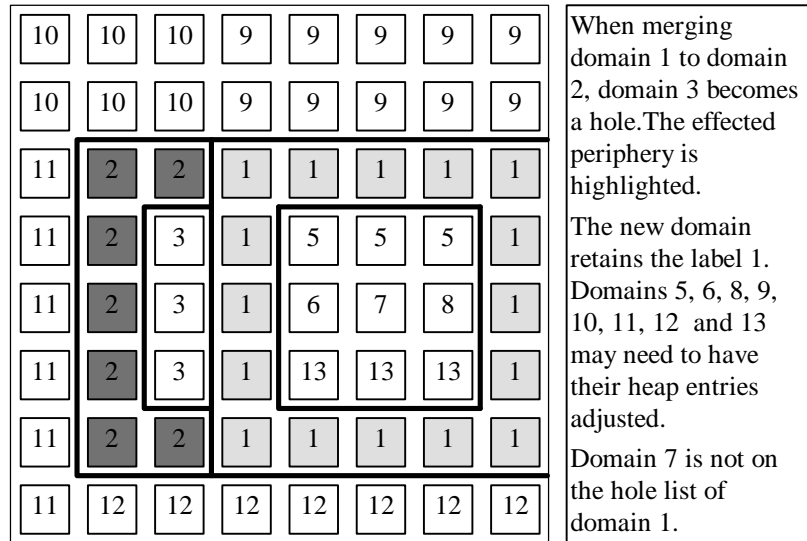


Figure 4.2  
Hole Discovery

A left turn takes place if neither of these two conditions are met.. A clockwise traversal is made by reversing the roles of left and right in the previous discussion.

#### 4.4.2.3 Hole Discovery

Figure 4.2 is an example of hole discovery. Hole discovery is the process of determining which neighboring pixels become holes when two domains are merged. The key idea is that a domain touched when traversing the outer boundary of either domain before their merger and not touched when traversing the outer boundary of the combined domain, must have become a hole of the combined domain.

As discussed previously, an accumulator is a fixed size set which is sized to hold the maximum number of possible domain neighbors (We discuss this maximum in Section 4.6). As neighboring domains are first encountered during a traversal, they are added to an accumulator. Subsequent encounters update previously existing accumulator entries. Each traversal has a unique identifier or age. Accumulator entries added or updated during a traversal have that traversal's age

Holes are discovered by traversing the boundaries of the domains being merged and accumulating neighbors that are on either boundary. After the domains are merged, the accumulator age is incremented and neighbors on the combined boundary are accumulated. Any neighboring domains encountered in the post-merger boundary traversal have their corresponding accumulator entries updated to the age of the post-merger traversal.

Following the post-merger boundary traversal, the accumulator is enumerated. Each entry in the accumulator (there is one for each domain encountered in any of the three boundary traversals) is retrieved and its age is examined. Any entries older than the post-merger traversal are new holes.

#### 4.4.2.4 Domain Tree Maintenance

When two domains are merged, holes in the original domains and those created by the merger must have a new best neighbor search performed. Each domain's hole set makes this process efficient. Each domain is either in the hole set of another domain or in the set of domains that are not interior to any other domain.

Initially, every domain is in the *global* domain set. As domains are merged and holes are formed, both new and old holes must be moved to the appropriate parent set. Holes that exist before a merge remain holes. The holes of the domain being subsumed are moved to the remaining domain's hole set. Newly discovered holes are added to the remaining domain's hole set.

If the domain being *subsumed* is in a hole set and the subsuming domain is not, the subsuming domain is removed from the global set and placed in the set of the subsumed domain. This can occur when the subsuming domain is completely surrounded by another larger domain but it does not actually touch the surrounding domain. This subsumed domain can be completely surrounded by intervening holes of the larger surrounding domain. If the domain being subsumed is the surrounding or parent domain of the subsuming domain, the new combined domain is placed in the hole set of the subsumed domain.

Build a heap by inserting one entry for each domain in a seed partition.

Loop until the desired number of domains is obtained:

- Remove the head of the heap.
- Extract its neighbor's heap entry.
- Merge the two domains (smaller to the larger):
  - If merging a parent into one of its holes:
    - Place the resulting domain into the set of the parent.
  - If merging a hole into an interior global domain:
    - Place the resulting domain into the set of the hole.
- Perform the merge:
  - Travel the neighbor's boundary and accumulate.
  - Travel the head's boundary and accumulate.
  - Advance the accumulator's age.
  - Call the merge callback routine.
  - Merge the pixel sets (smaller domain's tags change).
  - Add the subsumed domain's holes to the combined domain's hole list.
  - Travel the combined boundary and accumulate.
- Enumerate the accumulator:
  - If an entry is young (on the combined boundary):
    - Find the cost of merging the domain with the new combined domain.
    - Save as the best cost if better than the current best cost.
    - If the entry's heap entry points to either original domain:
      - Find the entry's best neighbor.
      - Update the entry's heap entry with best neighbor information.
  - Or for older entries (not on the combined boundary):
    - Move their domain to the combined domain's hole list.
- Insert an entry in the heap for the combined domain using the best cost.
- For each hole of the combined domain:
  - Find its best neighbor and update its heap entry.

End of loop

Figure 4.3  
Greedy Domain Extraction Algorithm

## 4.5 Proof of Correctness

### 4.5.1 Definitions

A heap entry contains *references* to a pair of adjacent domains. A pair of adjacent domain references is a *boundary*. An entry also contains the *cost* of merger of its domain pair. The more positive the cost associated with the merger of two domains, the more favorable the impact on the global cost function if the two domains are merged. The most positive entry is always at the head of the heap if the greedy order is maintained.

A *valid* heap entry has the following properties:

1. The boundary to which it refers is part of its associated domain's outer boundary.
2. The neighbor to which it refers has its own identity (It has not been merged with another domain) and is an actual neighbor of the heap entry's associated domain.
3. The cost of merger in the entry is the correct cost of merger with the indicated neighbor.

A boundary between two adjacent domains is *covered* if:

1. Neither domain is a hole of the other and
2. Either of the adjacent domains' heap entries contains the boundary, or at least one of the domains' heap entries contains a boundary that has a more positive cost.

or:

1. One domain is a hole of the other and
2. The hole's heap entry either contains the boundary or contains a boundary that has a more positive cost.

### 4.5.2 Overview

We first prove that a heap entry persists for each domain and that all the boundary of the partition remains covered by a heap entry. We then prove that all heap entries remain valid and that the best merge boundary is always in the heap. Finally, we prove that the best merge boundary is at the head of the heap.

### 4.5.3 Details

**Lemma 1.** Every domain in the partition has a corresponding merge heap entry.



The proof is by induction on the number of merge operations. By definition, when the heap is built, every domain has a corresponding heap entry. When two domains are merged, their corresponding heap entries are extracted. A new entry is added to maintain the invariant.

**Lemma 2.** All boundaries in the partition are covered by an entry in the merge heap.

The proof is by induction on the number of merges. Certainly, every boundary is covered initially. Since each entry in the heap contains the best merge candidate for its associated domain, every portion of its outer boundary is covered. Since all of the outer boundary of every domain in the partition is covered, every boundary in the partition is covered.

The only *boundary sections* that are effected by a merge are those that touch the merged domains. These sections are those on the outer boundary of the merged domain and those on the outer boundaries of the holes of the merged domain that touch the merged domain. The outer boundary of the merged domain is covered by the new entry inserted in the heap after the merge. Since the heap entries for each hole of the combined domain are adjusted after the merge, the hole boundaries remain covered.

**Lemma 3.** The boundary with the most positive cost in the partition is in the heap.

The proof is by contradiction. Suppose that the best boundary were not in the heap. By Lemmas 1 and 2 every domain in the partition has a corresponding heap entry and all boundaries are covered by entries in the heap. If the boundary is between two domains that are not holes of one another, at least one of the adjacent domains' heap entries must refer to a boundary with a more positive cost. Since this boundary would have a more positive cost, and is in the heap, the best boundary must be in the heap. If the boundary is between a hole and its parent, the hole's heap entry must either contain the parent boundary or a boundary that has a more positive cost. Again, a contradiction.

**Lemma 4.** All entries in the heap remain valid.

The proof is by induction on the number of merge operations. If after  $N$  merges the heap contains all valid entries, it does so after  $N+1$  merges:

- The only entries that could be effected by a merger are those associated with domains that abut either to the interior or to the exterior the two merged domains.
- Any heap entry for a domain that touches the combined outer boundary and contains as its neighbor member either of the two prior to combination domains is adjusted after the merge operation.
- The heap entries for all holes of the combined domain are adjusted after the merger and therefore remain valid.

**Lemma 5.** The head of the heap contains the merge entry with the most positive cost.

By lemma 3 the most positive cost merge entry is in the heap. By definition of the operations of insert, extract head, extract, and adjust, the head of the heap contains the entry with the most positive cost.

**Theorem 1.** Domains are merged in the proper greedy order.

The proof is by construction. The merge heap remains valid by Lemma 5. The head of the heap contains the best merge candidate by Lemma 4. Every boundary in the partition remains covered by Lemmas 1, 2, and 3.

#### 4.6 Algorithmic Complexity

The operations performed during a merge fall into four categories: set tag maintenance, heap maintenance, boundary traversal, and external operations performed by the boundary traversal, domain merger, and cost function callback routines. The domain merger routine is called once for every merger. The boundary traversal callback routine is called once for every boundary separator along the external and internal periphery of the merged domains. The cost function routine is called once for every domain touching the same periphery. For this analysis we assume that all three callback routines are  $O(1)$ . This requirement calls for careful construction of the domain model.

Given an image  $\Phi$ , let  $N = |\Phi|$  designate the number of its pixels. If  $T$  is the target number of domains in a partition of  $\Phi$ , then the number of merges necessary to reach  $T$  is  $M$ . If  $T \ll N$ , then  $M \rightarrow N$  as  $T \rightarrow 0$ .

Since when domains are merged, the smaller domain is merged to the larger, the total number of membership tag changes for  $N$  merges is  $O(N \log N)$ <sup>13</sup>. Additionally, each merger results in a boundary traversal and potential cost function,  $O(1)$ , and heap operations,  $O(\log N)$ , for each adjacent domain. Unfortunately, the number of potential boundary locations and the number of neighboring domains encountered during a traversal is  $O(N)$ . For  $N$  merges, the overall complexity is:

$$O(N \log N) + O(N) \cdot (O(N) + (O(1) + O(\log N)) \cdot O(N)) = O(N^2 \log N).$$

where the complexity terms are for membership tag changes, boundary length traversed (including boundary callbacks), bulk callbacks, and heap operations respectively.

Although the length of the periphery of a domain is  $O(N)$ , this bound is not often approached in practice. Let  $|B_e(D)|$  designate the length of the external boundary of a domain, and  $|B_h(D)|$  designate the total boundary length of all the holes in a domain. If the following restrictions are placed on the domains of  $\Phi$ :

Largely Convex

$$\exists c_1: \forall D \in \Phi, |B_e(D)| < c_1 \sqrt{|\Phi|}$$

Largely Whole

$$\exists c_2: \forall D \in \Phi, |B_h(D)| < c_2 \sqrt{|\Phi|}$$

Then the overall complexity reduces to:

$$O(N) \cdot O(\sqrt{N}) \cdot O(\log N) = O(N \cdot \sqrt{N} \cdot \log N).$$

#### 4.6.1 Dynamic Size Limiting

The complexity bound can be reduced still further by placing a dynamic limit on the maximum domain size. For instance, when a large number of domains remain in the partition, the maximum domain size can be kept smaller than when a small number of domains remain. A convenient bound (from a complexity perspective) to place on the size of a domain is  $c \cdot 2^{\lfloor \log_2 \frac{N}{Z} \rfloor}$ , where  $Z$  is the number of domains remaining.

Dynamic size limiting divides the original problem into  $\log_2(N)$  subproblems that can be summed thusly:

$$\sum_{T=0}^{\log_2 N - 1} \frac{O(N)}{2^{T+1}} \cdot O\left(\log_2 \frac{N}{2^T}\right) \cdot (c \cdot 2^T).$$

The first term of the sum is the number of merges in the subproblem, the second term is the complexity of heap operations for each merge in the subproblem and the third term is the complexity of the boundary and neighboring domains for each merge in the subproblem. After taking  $N$  outside and canceling terms, this reduces to:

$$\frac{c}{2} \cdot O(N) \sum_{T=0}^{\log_2 N - 1} O\left(\log_2 \frac{N}{2^T}\right).$$

Since the sum is bounded by  $O(\log^2 N)$ , the overall complexity reduces to:

$$O(N \log^2 N).$$

## 4.7 Summary

We presented a new greedy domain growing algorithm for image model extraction. The algorithm is independent of the exact image model which may contain both bulk and boundary components. The model is updated via callback routines as the algorithm progresses.

We initialized the algorithm by creating one domain for each image pixel. We used a heap to maintain the greedy ordering and developed update and extraction methods to aid in heap maintenance. We developed the boundary traveler and showed how local boundary

characteristics are re-measured after each domain merger. We kept track of neighboring domains of a boundary traversal via a fixed size accumulator. We showed how the algorithm handles inclusions or holes which may form from the merger of two sibling domains or disappear via merger with a parent.

We proved correct greedy order through the idea of boundary covering. A complexity bound of  $O(N \log^2 N)$  was developed by dynamically increasing maximum domain size as domain count falls.

## 5. Domain Smoothing

### 5.1 Motivation

Once an image has been partitioned into a set of domains it is often desirable to refine the domain boundaries. For example, the boundary may be somewhat irregular or “noisy” due to imperfections in the extraction procedure or noise in the underlying image. An irregular boundary is quantitatively different from a smooth boundary in two ways. It is longer and is more random. Since the primary motivation of partitioning a digital image is most likely to be to simplify its description, both of these properties are undesirable in that they lengthen the description of the partition. Irregular boundaries have another intangible property that makes them undesirable: they are not “visually pleasing”. With a proper smoothing procedure, one can both shorten the description and produce a more visually pleasing partition.

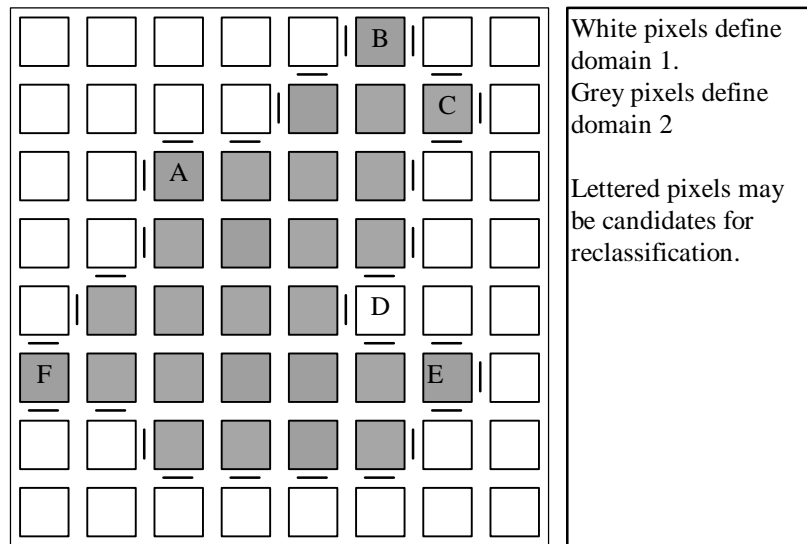


Figure 5.1  
Irregular Boundary

In Figure 5.1 the gray domain contains only 27 pixels, but the boundary between it and the white domain is 32 separators long. In addition the boundary has little structure in that if one follows the boundary it is difficult to predict what direction it will next take using only information about what direction it has taken thus far. Intuitively, the pixels labeled with

letters seem to be the loci for much of the boundary's length and irregularity. For example, if pixels E and D were reclassified, as white and gray respectively, the boundary length would be reduced by four and become more regular. The rest of this chapter develops a formal procedure for making such reclassifications.

## 5.2 Background

There is one body of significant work in boundary relaxation that originates with the Gibbs field paper of Geman and Geman<sup>7</sup>. After introducing a so called edge process embedded in a Gibbs field, certain local boundary configurations or cliques were assigned a more favorable energy and this additional edge information was used to improve the results of a simulated annealing image restoration algorithm. Variations of this technique have been used for image segmentation via region growing and for boundary relaxation. A recent example is that of Hussain and Reed<sup>14</sup> who used the technique in a segmentation based image compression method.

Another technique that has developed a significant following is the method of Snakes<sup>15</sup>. In this method a tentative boundary, or Snake, is optimized by balancing an attraction to image features with an internal energy related to spline continuity. It seems especially suited to interactive specification of image contours due to its sensitivity to the position of the initial tentative boundary.

## 5.3 Boundary State Machine

While previously reported methods are quite powerful, they seem somewhat ill-suited to the task of efficiently and autonomously smoothing domain boundaries. If we seek to develop a fast deterministic procedure, we first might ask: What is the asymptotic limit? An obvious limitation is that each boundary pixel must be visited at least once. If after one move is performed, the number of pixels revisited to determine the next desirable move is limited to a constant times the number of pixels just moved, the number of pixels moved must be counted. Defining  $B$  to be the number of boundary pixels and  $M$  to be the number of pixels moved, the lowest possible algorithmic complexity is:

$$O(B + M).$$

To achieve this limit the initial pixel search set must be limited to an integer multiple of the total boundary length. One way to do this is with a state machine that traverses the boundary between domains.

In a boundary traversal, we follow the boundary as one would follow a maze. For every traversal there is a *home* domain. At each pixel corner a direction decision is made: either turn left, right, or continue straight. The path traversed follows the boundary of the home domain without diverging. We require that all home domains be connected so that a traversal is always *closed*. That is, it always returns to the starting point without doubling back.

A boundary separator is *labeled* with the event associated with the immediately preceding pixel corner. For example, on Figure 5.1 starting at the boundary separator above pixel C and traveling in the counterclockwise direction, the first ten movements are: *rlrlrslr*. The counterclockwise direction of travel is the default in the rest of the discussion, but using mirror symmetry simply interchange right and left when reversing direction.

The last few turn decisions make up the machine's state. Using regular expression syntax, we can encode the state at the edge just below pixel E in Figure 5.1 as  $ls^+lr$ . Formalizing this procedure, the oldest event is written on the left and more recent events follow to the right. In this example we have limited the state to 4 symbols, but we may expand that as necessary. We now turn to the task of finding boundary states that can be used to perform smoothing.

#### 5.4 Rasterization

While boundary length is a significant factor in measuring edge smoothness, it cannot account for all cases where a boundary may be considered irregular. For example the pixel labeled A in Figure 5.1 is visually perceived as a “bump” on a diagonal line but moving it to domain one does not change the overall boundary length. We now use a well known result from computer graphics<sup>16</sup> to develop a more powerful smoothness measure:



*The resulting smoothed boundary should contain the smallest number of raster drawn lines and curves consistent with the underlying pixel intensity approximation function. Raster lines and curves have the simple property that they shift at most one column at a time if the absolute value of their slope is greater than or equal to 1 and at most one row at a time otherwise.*

Using the raster concept combined with the boundary state machine developed previously, we can now make the following definitions:

- A *raster section* is a contiguous set of movements that obeys the rules for raster drawn features.
- A *left straight section* is a contiguous straight section immediately following a left turn.
- A *bump* is a left turn following a left straight section.
- A *straight section* is a non-null set of contiguous straight movements.
- An *odd turn section*, abbreviated *o*, is contiguous odd number of alternating turns.
- An *left turn section*, *o<sub>l</sub>*, is an odd turn section beginning with a left turn.
- An *right turn section*, *o<sub>r</sub>*, is an odd turn section beginning with a right turn.
- A *corner* is a straight section followed by a left turn section followed by another straight section.
- A *dimple* is the right turn bump.
- An *indent* is the right turn corner.
- A *raster break* occurs at the last turn of a bump, corner, dimple or indent.

## 5.5 Boundary Cost Function

Before developing some smoothing transformations, let's first assign a cost to each transformation. The cost function has two major components, the boundary term and the underlying model error term. If we assume, that before smoothing, the boundary is generally in a favorable position relative to the underlying model, a smoothing transformation will most likely increase the error between the model and the source image. We can think of a smoothing transformation as increasing the smoothness of the boundary at the expense of increasing the underlying model error.

If the following definitions are made:

- $\Delta l$  - The reduction in number of separators in the boundary
- $\Delta b$  - The reduction in the number of raster breaks in the boundary
- $\Delta E$  - The increase in error in the underlying intensity function model

the cost associated with a boundary transformation can be formulated thusly:

$$\Delta C = -\Delta E + \alpha \cdot (\beta \cdot \Delta l + \delta \cdot \Delta b),$$

where  $\alpha$ ,  $\beta$ , and  $\delta$  are arbitrary constants. In this formulation,  $\Delta l$  and  $\Delta b$  are positive for a decrease in boundary length or raster break count respectively. Any decrease in boundary noise is played against any increase in model error. The more positive the cost function, the more favorable the outcome of the boundary transformation.

The raster criteria allows us to split the boundary component of the cost function into two terms, one proportional to the boundary length and another proportional to the number of raster breaks. If a transformation is only made if it results in a positive  $\Delta C$ , the parameter  $\alpha$  controls the degree smoothing.

## 5.6 Smoothing Transformations

Given that reducing the number of boundary separators and raster breaks produces a smoother boundary, we can use certain states in our boundary follower as *transformation markers*. As discussed previously, the signaling boundary features are the bump, corner, dimple and indent. If we assume that we traverse each boundary in both directions, it is only necessary to account for either the left or right handed features. We arbitrarily choose left handed features for the rest of the discussion. Using regular expression notation, Table 5.1 shows the bump and corner transformation markers.

Bump	$l^*l$
Corner	$sl(rl)^*s$

Table 5.1  
Transformation Markers

A transformation marker exists at every boundary locus where it is possible to move a connected set of pixels from one domain to another and reduce the number of separators or raster breaks. If we assume a counterclockwise direction of travel and left handed marker features, the pixels to be moved, or *move set*, are those under the bump or inside the corner.

Whether or not a pixel is moveable and other details of the move set are discussed in Sections 5.9 and 5.10. For now, we assume that only two domains are involved in each proposed transform and that all relevant pixels are moveable.

### 5.6.1 Move Sets

On Figure 5.1 the pixels labeled with letters are examples of single pixel move sets. Move sets are not restricted to a single pixel. The number of pixels in the move set is the product of the *length* and *depth* of the set. The length of a bump move set is one plus the number of straight events in the bump marker. The length of a corner move set is the number of left turns in the corner marker.

The depth of a move set is dependent upon the boundary *context* (state) in which the transformation marker is embedded. The context immediately preceding a transformation marker is called the *prefix* and that following is called the *postfix*. The number of immediately contiguous straight events in the prefix and postfix determine the move set depth. If the straight sections in the prefix and postfix are of different length, the shortest is the *controlling straight length*. The depth of a move set is one plus the controlling straight length.

Note that the corner transformation marker includes one straight section at each end. This straight section is not counted in the corner depth calculation.

Figure 5.2 shows bump and corner markers with length and depth of two. The arrows indicate the direction of movement. For example, if the arrows point to the right, the black pixels are to be moved from their current domain to the domain of the pixels just to their right. Depth is counted parallel to the movement axis and length perpendicularly.

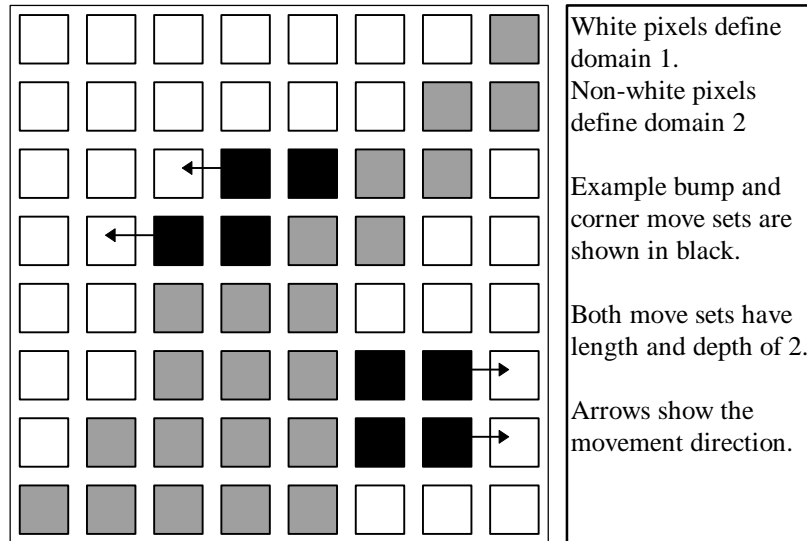


Figure 5.2  
 Move Sets

### 5.6.2 Limiting Move Depth

Up to this point, we have placed no limit on the size of a transformation move set. Intuitively, the larger the area of a transformation marker, the lower the likelihood of a favorable cost outcome for the transformation. Since large area transformations are quite unlikely and computationally expensive to evaluate, it is convenient to limit the move set depth to keep the computation efficient.

It is also desirable to be able to *partially reduce* a transformation marker. A partial reduction is a reduction of the depth of the marker. A depth reduction does not change a marker to one of a different type. A reduced marker retains its type if its depth before reduction was greater than one.

Since a reduced marker can be completely transformed simply by making multiple passes through it, a natural simplification is to limit the move set depth to one. Multiple passes peel away each transformation marker as layers of an onion. On

Figure 5.2 the pixels immediately adjacent to the arrows are members of the depth one, or *string*, move sets.

## 5.7 Determining $\Delta b$

A transformation marker's prefix and postfix determine its reduction in raster breaks,  $\Delta b$ . Contexts that produce a non-zero  $\Delta b$  are called *active*; others are *inactive*. An active prefix's mirror image is an active suffix. An *isolating* prefix ends with  $s$  and an isolating postfix begins with  $s$ . Markers of depth two or greater have both an isolating prefix and an isolating postfix and are obviously inactive.

If a marker feature is deeper than one, its move set is isolated and  $\Delta b$  resulting from a transformation is trivially zero. If a marker feature has depth one, its transformation may or may not result in a reduction in the number of boundary raster breaks. For example, a bump in the context  $sr/sr/s^*/lr/sr/s$  reduces to  $sr/s^*/lr/s$  and there is no net change in  $b$ , but a bump in the context  $rsr/s^*/lr/sr$  reduces to  $rsr$  for a net reduction of two raster breaks. Clearly,  $rsr$  is an example of an active prefix and  $sr/$  is an example of an inactive prefix.

Active contexts generally must contain a mirror element of their transformation marker. Odd turn sections are useful for absorbing the core of the corner transformation marker, which happens to be an odd turn section itself. Any corner transformation that reduces the raster break count must have an opposing odd turn section as a prefix or postfix. Odd turn sections as prefixes and postfixes are independent. If both are present  $\Delta b$  is twice as large as if only one were present.

Marker	Postfix	$\Delta b$
Bump	$rs^+o s^+$	1
Bump	$rs^*r$	$2^{\pm}$
Corner	$o_r s$	1

<sup>±</sup>In either or both positions.

Table 5.2  
Transformation Marker Active  
Postfixes

Bump transformations are more complex than corner transformations. The opposite of a bump is a dimple. A dimple must be a prefix or postfix for a bump transformation to reduce the number of raster breaks. Unlike odd turn sections, bump prefixes and postfixes are not cumulative.  $\Delta b$  is identical if either one or both are present. Additionally, a bump marker with the proper surrounding context may also include up to two indents. The maximum  $|\Delta b|$  occurs when a bump marker has an indent prefix and dimple postfix or vice versa.

### 5.8 Determining $\Delta l$

It is desirable to define the boundary length,  $l$ , in a manner that associates a smaller length (lower cost) to a partially reduced transformation marker. The conventional method of defining boundary length is to count separators, which is fine for bump transforms but inadequate for corner transforms.

If pixel E on Figure 5.1 was moved from the gray to the white domain, the number of separators in the boundary would be reduced by two. In this case, the separators to the east, north, and south of pixel E would be replaced by one to the west. It is clear that for a bump transform involving only two domains, the number of separators is always reduced by two.

Pixel A on Figure 5.1 is an example of the simplest corner transform move set. If pixel A were transferred from the gray to the white domain, the separators to the north and west would be replaced by separators to the south and east. This result generalizes to all corner transformations. The net change in the number of boundary separators is zero.

### 5.8.1 Diagonal Separators

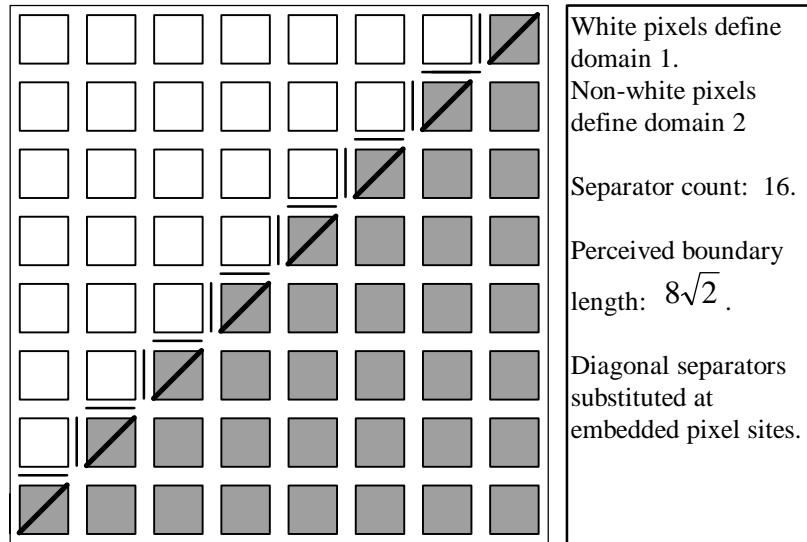


Figure 5.3  
 Perceived Boundary Length

The asymmetry in the conventional boundary length accounting for bump and corner transforms is due to an artificially high cost placed on diagonal lines. A diagonal line in our four direction boundary consists of pairs of alternating turns. In the context of the diagonal boundary on Figure 5.3 each turn pair has a perceived boundary length of  $\sqrt{2}$ .

Conceptually, if a pixel is *corner embedded* in only two domains we can replace the horizontal and vertical separators abutting it with a single diagonal separator. If we approximate the  $\sqrt{2}$  perceived length of diagonal separator with 1.5, we can retain integer weights simply by scaling all costs. Our new *conceptual boundary* has two types of separators. We designate diagonal separators with  $d$  and rectilinear separators with  $r$ .

### 5.8.2 Boundary Length Accounting

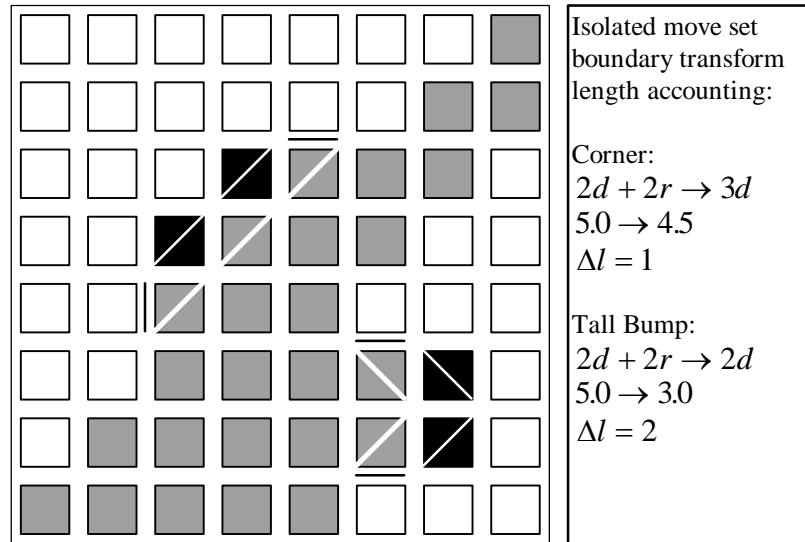


Figure 5.4  
Boundary Length Accounting

Figure 5.4 shows example boundary length accounting for example corner and bump transformations. Boundary length calculations are performed against the conceptual boundary previously defined. Diagonal separators are assigned a weight of 1.5 and rectilinear separators are assigned a weight of 1. The example bump transformation shown is a *tall* bump. A tall bump has straight events at the beginning of its postfix and at the end of its prefix. A *short* bump is surrounded by right turns. A *mixed* bump has a turn adjoining in the prefix and a straight event adjoining in the suffix or vice versa.. A *tower* bump has no central straight events. Table 5.3 summarizes  $\Delta l$  for transformation marker types and subtypes. Marker length is designated  $m$  in the table.



Marker	Type	Separator Mutations	$\Delta l$
Bump	Shallow	$(m - 2)r + 2d \rightarrow mr$	1
Bump	Mixed	$(m - 1)r + 2d \rightarrow (m - 1)r + d$	1.5
Bump	Tall	$mr + 2d \rightarrow (m - 2)r + 2d$	2.0
Bump	Tower	$3r \rightarrow r$	2.0
Corner	All	$md + 2r \rightarrow (m + 1)d$	0.5

Table 5.3  
Transformation Boundary Length Changes

## 5.9 Convergence

Up to this point we have assumed that only two domains are involved in a boundary transformation. If this requirement is met, it is clear that the algorithm must converge since after every transformation the conceptual boundary length is reduced. When more than two domains about a transformation, more analysis is needed.

### 5.9.1 Interfering Domains

When the pixels of a transformation marker touch more than two domains, at least one of the domains is an *interfering domain*. The home domain is by definition a non-interfering domain, so interfering domains are always neighboring domains.

On Figure 5.5, the gray domain is the home domain. Domains C and D are *receiving* domains. A receiving domain is the domain to which moved pixels are transferred. Every transformation has one home and one receiving domain. The receiving domain must touch every pixel of a move set in the direction of transfer.

Domains A, B, and E are interfering domains. Domain B causes *splitting* interference. Its presence splits the destination of the move set into two domains. This has an effect on domain boundary accounting but it is precluded in the present algorithm simply because it increases the complexity of domain management. It is possible to account for the presence of splitting interference in boundary convergence accounting, but we do not do so.

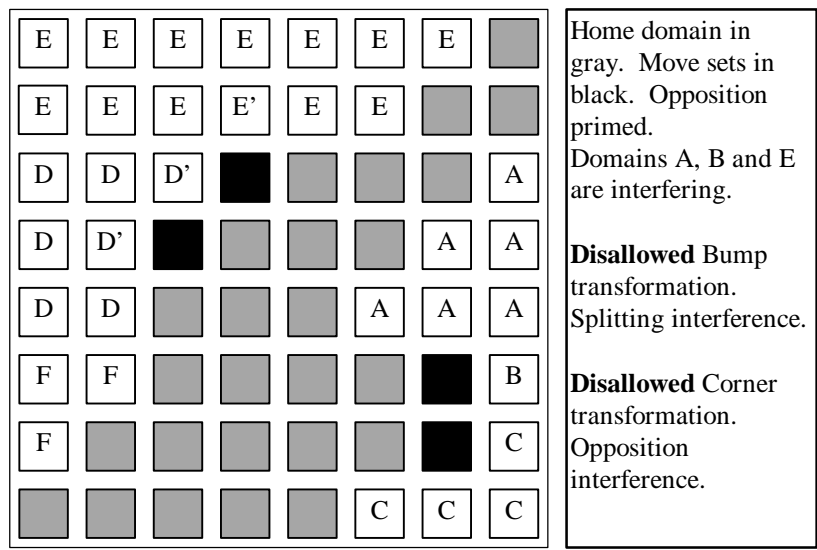


Figure 5.5  
Interfering Domains

Domain E is an *opposition* interfering domain. Opposition interference is indicated when a one of the pixels of a corner transformation move set is not corner embedded and a conceptual diagonal separator cannot be formed. Like splitting interference, opposition interference can be accounted for in  $\Delta I$  calculations, but is precluded in the present algorithm. The primed pixels in Figure 5.5 are the *opposition set* of the black corner move set. The opposition set of a corner transformation marker must lie in a single domain or the transformation is not performed.

5.9.2 *Subtractive Interference*

Domain A is a *subtractive* interfering domain. Its presence decreases the amount of boundary smoothing produced by the transformation. A bump transformation marker can have a subtractive interfering domain at each of its ends. A subtractive interfering domain reduces the  $\Delta I$  of a marker end to zero. We have handled splitting and opposition interference by precluding moves where they occur. We will not preclude moves with subtractive interference, however, and we must take care to assure that it does not effect convergence.

Previously, we proved that when all transformations were non-interfering, the algorithm converged to a fixed point. The key to the proof was a requirement that  $\Delta l$  be negative for all transformations. We relax that requirement for transformations with subtractive interference.  $\Delta l$  for such transformations must be non-positive. However, the requirement for  $\Delta E$  is tightened. When  $\Delta l$  is zero,  $\Delta E$  must be negative.

Since at any given total boundary length transformations that leave the boundary length unchanged must reduce the model error, the algorithm is guaranteed to converge at any total boundary length. Since no transformation is allowed to increase the boundary length, the algorithm must converge at the smallest attained boundary.

### 5.9.3 Summary of Convergence Requirements

- Transformations with splitting interference are disallowed.
- Transformations with opposition interference are disallowed.
- Transformations with subtractive interference must have negative  $\Delta E$

## 5.10 Correctness

Before smoothing, the domains of our image partition are connected. Connectedness is defined as follows:

- *For a connected domain, there is a path between the centers of any two pixels in the domain that makes only vertical and horizontal moves and that only visits pixels in the domain.*

The smoothing algorithm must maintain connected domains. To do this we introduce the notion of a *connection hull*. On Figure 5.6 move sets for a corner and bump transformation marker are labeled with M. The connection hulls for the example transformation marker are labeled with C.

Before precisely describing the connection hull, let's state the first correctness requirement.

- *The connection hull of a transformation marker must lie completely within the marker's home domain.*

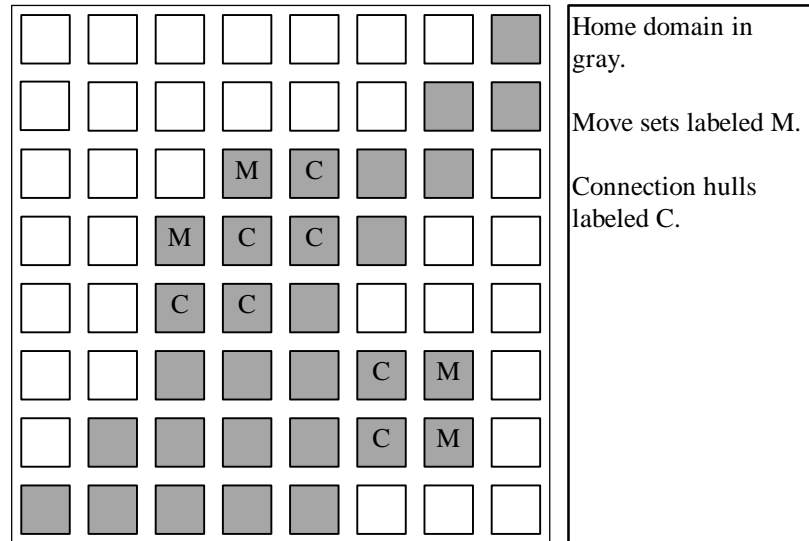


Figure 5.6  
Connection Hulls

The connection hull is slightly different for the two types of transformation markers and can be described precisely once we make the following definition. The *neighbor set* of a transformation marker includes the pixels touching its move set along edges that are not part of the marker.

- *The connection hull of a bump transformation marker includes the pixels of its neighbor set that touch its move set in the direction opposite to that of movement.*
- *The connection hull of a corner transformation marker includes its neighbor set and the pixels not in its move set that touch its neighbor set along at least two edges.*

As can be seen from Figure 5.6 the connection hull consists of the pixels beneath the transformation marker. The purpose of the connection hull is to provide a path around the pixels of the move set. If the hull is not complete, the transformation is disallowed.

### 5.11 Computational Complexity

Obviously we must traverse every boundary site at least once. Since the move depth is limited to one, this traversal is  $O(B)$ , where  $B$  is the total boundary length of the partition. Each move transformation is  $O(|m|)$ , where  $|m|$  is the number of pixels of the move set. This bound is only possible if we use an  $O(1)$  method like that of Chapter 3 for determining the

error change in the underlying model. If the sum of  $|m|$  for all move transformations is  $M$ , the complexity of all move transformation is  $O(M)$ .

Once a transformation is made, we must re-traverse the effected boundary to see if another transformation should be made. The total re-traversal is also  $O(M)$ . The overall algorithmic complexity is

$$O(B + M).$$

### 5.12 Limitations

When a transformation marker is encountered during a boundary traversal, it is not possible to know if making a transformation reduces the number of raster breaks,  $b$ , in the boundary without first looking for an active postfix. A smoothing algorithm that is *causal* is computationally preferred. The complexity of the state machine increases fairly quickly if all cost function parameters cannot be determined at the point at which the transformation marker is first discovered. There does not seem to be any way to avoid non-causality and still use  $\Delta b$  as a cost function term. Fortunately, experiments have shown that the algorithm exhibits very good performance when  $\delta$ , the arbitrary weighting placed on  $\Delta b$ , is set to zero.

The algorithm's key limiting constraint is the inability to increase the boundary length if doing so would produce a better match to the underlying model. If, for a given domain, all of the unsmoothed boundary lies within the optimal smoothed location, the algorithm's result also lies within the optimal location. A key observation is that no part of the boundary can move horizontally or vertically beyond the locations of the initial unsmoothed horizontal and vertical maxima and minima.

### 5.13 Experiments

We now present the results of using the smoother on one synthetic and one natural image. We use the following parameters

$$\begin{aligned}\Delta E &= \Delta MSE \\ \delta &= 0\end{aligned}$$

$$\beta = 1$$

with the smoothing cost function:

$$\Delta C = -\Delta E + \alpha \cdot (\beta \cdot \Delta I + \delta \cdot \Delta b)$$

defined in Section 5.5.

Since  $\delta = 0$ , the raster break count is not used in the cost function and it reduces to:

$$\Delta C = -\Delta E + \alpha \cdot \Delta I$$

The results without  $\Delta b$  are quite good and  $\Delta b$  calculation has been deferred to future work. Results for one synthetic image, Syn2, and one natural image, Lena follow.

Figure 5.7 is the synthetic image Syn4. Syn4 consists of three constant patches laid over a vertical ramp intensity function. The entire image is overlaid with additive noise,  $\sigma = 32$ . The average gray levels for domains one - three are 160, 224 and 64 respectively. With the origin in the upper left and coordinates increasing down and to the right, domain number four's intensity function is  $\Phi = y$ .

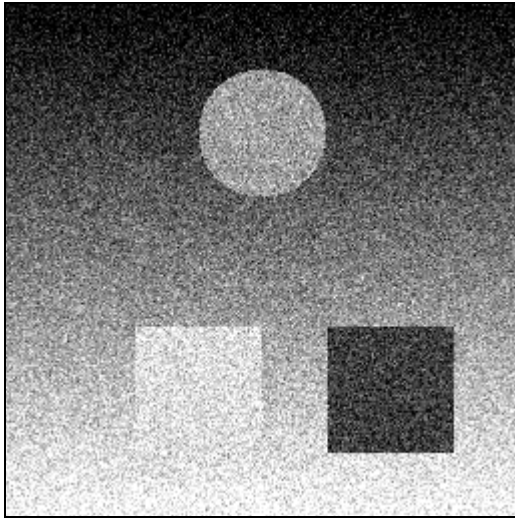


Figure 5.7  
Synthetic Image Syn4

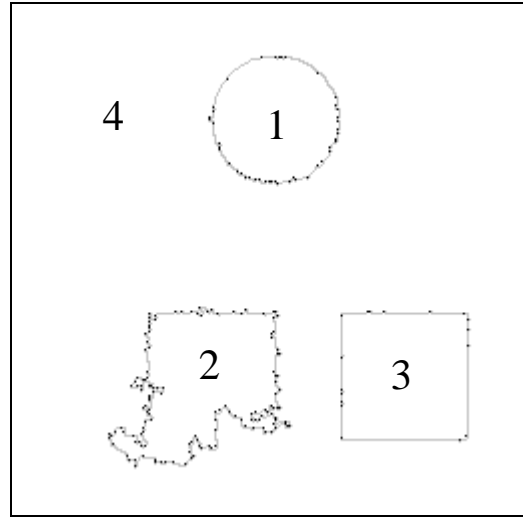


Figure 5.8  
Four Domain Partition of Syn4

Syn4 is designed to be quite difficult to partition. The domain extraction algorithm of Chapter 0 produced the domains of Figure 5.8. The signal to noise ratio at the lower boundary of domain 2 is  $-\infty$ , and indeed the extraction procedure has problems with this boundary. If the noise suppression factor is too large, domain 2 becomes too small. If it is too small, it spreads over the bottom of the image. Figure 5.8, corresponding to a suppression factor of 768, is the best result attainable.

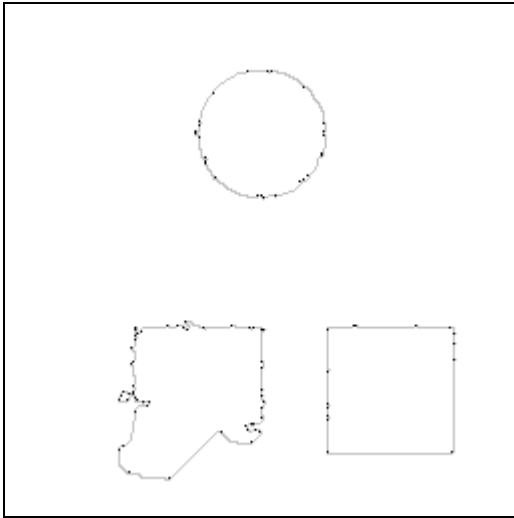


Figure 5.9  
Smoothed Syn4,  $\alpha = 256$

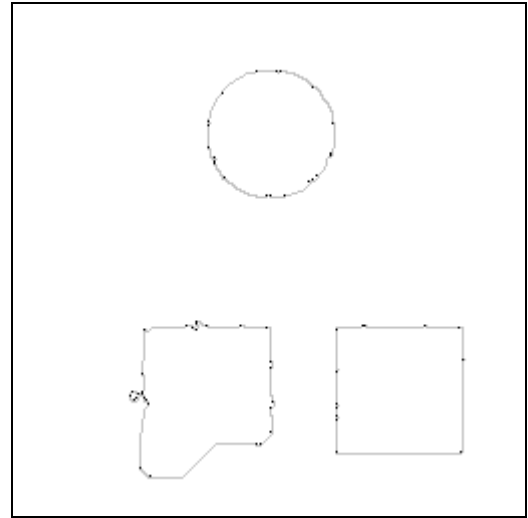


Figure 5.10  
Smoothed Syn4,  $\alpha = 512$

The surrounding figures show the result of smoothing the domain boundaries with increasingly larger smoothing parameter,  $\alpha$ . For lower values of  $\alpha$  (Figure 5.9, Figure 5.10) smoothing is highly local.

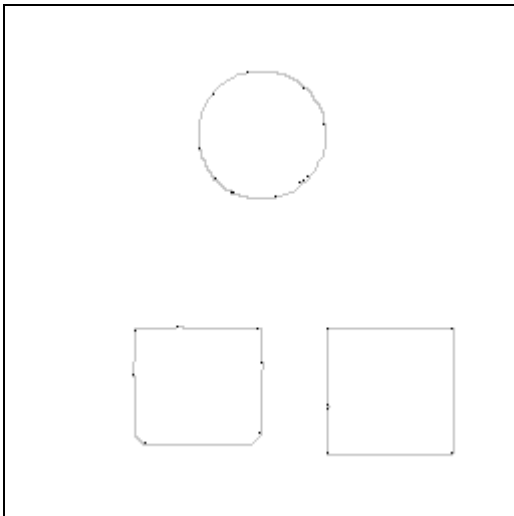


Figure 5.11  
Smoothed Syn4,  $\alpha = 1024$

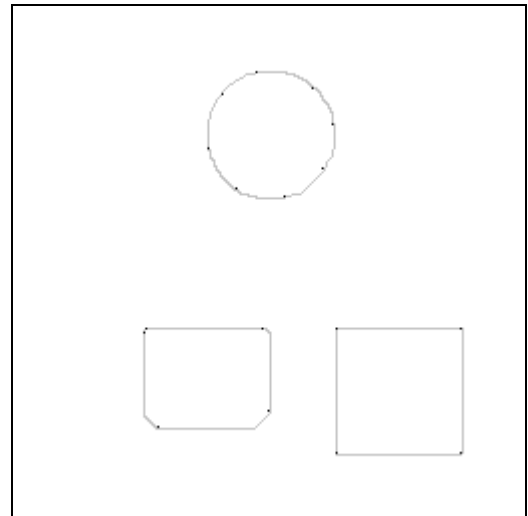


Figure 5.12  
Smoothed Syn4,  $\alpha = 8192$



As  $\alpha$  increases the degree of smoothing increases. The best results are attained for  $\alpha = 1024$ . The number of raster breaks is finally reduced to the number in the underlying image at  $\alpha = 8192$ .

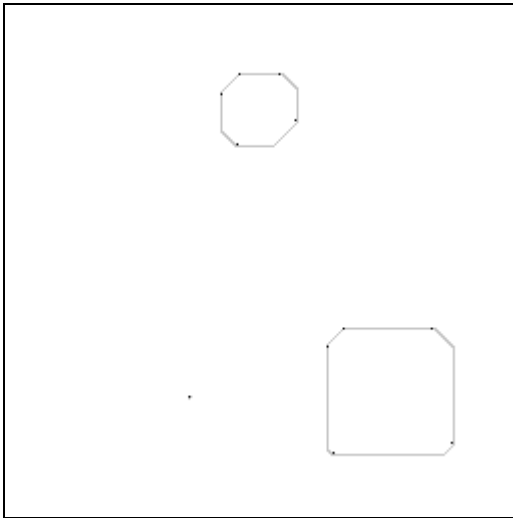


Figure 5.13  
Smoothed Syn4,  $\alpha = 100000$

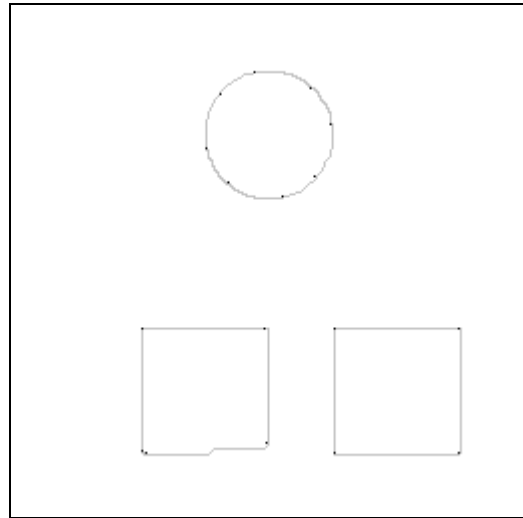


Figure 5.14  
Smoothed Syn4, Restorative mode,  $\alpha = 1$

Figure 5.13 shows how the algorithm converges to fixed points. One domain has collapsed completely and the circle is significantly distorted. Increasing  $\alpha$  further results in three fixed points.

Figure 5.14 shows the results of smoothing the initial partition against the image Syn3 (Syn4 without additive noise). The smoothing factor is set to one for this restorative experiment. The misplacement of the bottom edge of domain 3 is quite small. One half of the boundary is one raster row too low and the other half is one raster row too high.

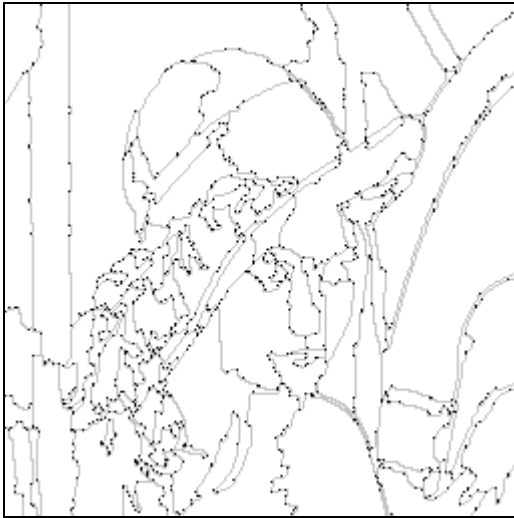


Figure 5.15  
Unsmoothed 100 Domain Lena

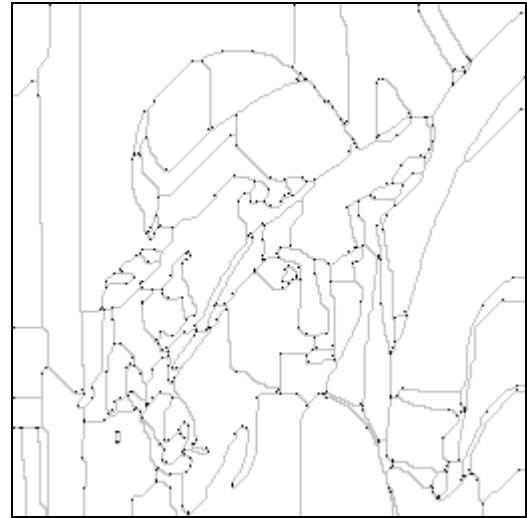


Figure 5.16  
Smoothed 100 Domain Lena,  $\alpha = 1024$

Figure 5.15 shows the unsmoothed domain boundaries for one test partition of Lena with 100 domains. Figure 5.16 is the same partition smoothed with a smoothing factor,  $\alpha$ , of 1024. A total of 855 transformations were performed to produce Figure 5.16. Raster breaks are shown as black dots in both figures.

<b>Lena</b>	<b>Boundary Separators</b>	<b>Raster Breaks</b>	<b>PSNR</b>	<b>MSE</b>
Unsmoothed	7946	946	27.4	118.5
Smoothed $\alpha = 1$	7525	572	27.8	106.8
Smoothed $\alpha = 1024$	6182	296	27.1	127.3

Table 5.4  
100 Domain Lena, Boundary Data

Table 5.4 is boundary data for the unsmoothed partition and smoothed partitions with two different values of  $\alpha$ . When  $\alpha$  is small the error in the underlying model actually decreases and only increases slightly with increased smoothing. The boundary length and number of raster breaks are decreased proportionately as smoothing increases.

#### 5.14 Summary

We developed the raster-break as a formal measure of boundary noise in an image partition and used it to design a state-machine boundary smoothing algorithm. We applied the moment

operators and error functions of Chapter 3 to the smoother's cost function. We showed the computational complexity of the smoother to be  $O(B + M)$  where  $B$  is the boundary length of the partition and  $M$  is the number of pixels moved.

## 6. Domain Boundary Coding

### 6.1 Background

The problem of coding domain boundaries in digital images has been addressed previously from two perspectives. *Chain coding*<sup>17</sup> is a method of coding a contour in the image plane with a series of movements (north south east west or right left forward backward) along the contour. *Raster neighborhood coding*<sup>18</sup>, scans a digital image in raster order and uses classification information available from previously decoded pixels (above and to the left of the current pixel) to develop a *context dependent probability estimation* of the *classification* of each pixel. This estimation is used to reduce the information needed to make a classification. Once all pixels have been classified, a boundary description can then be inferred by placing a boundary element between any two pixels with differing classifications.

Most work with contour coding has been on *black/white images*. We are interested, however, in coding the boundaries between domains in a partitioned *general* (four or more classifications) image. As we next show, there are subtle but important differences between contours on a black/white image and those on a partitioned general image.



#### 6.1.1 Chain Coding

Chain codes can be divided into two major families, the distinguishing feature being the number of possible directions of movement at each point in the chain. With the recent application of context dependent probability estimation to four-way chains<sup>19</sup>, the desirability of eight-way chains is somewhat attenuated. We focus exclusively on four-way chains.

Three types of information must be provided for a general-purpose boundary chain code: chain starting points, chain direction information, and chain termination indicators. When chain coding the boundary of black/white images<sup>19</sup>, each chain is guaranteed to return to its starting point. Obviously, no two black features can touch each other or they would be the same feature. This characteristic allows *self-terminating chains*, where termination information is implicitly delivered by return to the starting point.

Reliable termination conditions without backtracking allow for simplified direction information. Only three possible decisions are possible at each point on the boundary, turn left, right or go straight. The base information per chain event is reduced from two to  $\log_2(3)$  bits. Unfortunately, when chain coding the boundaries of a general image partition, things get more complicated and some form of redundancy must be introduced.

One way to deal with the termination problem is to completely traverse the boundary of all domains in a partition<sup>20</sup>. Unfortunately, this results in the traversal of all boundary separators *twice*. For example a vertical separator is encountered once when traversing the domain to its left and again when traversing the domain to its right.

Another possible solution is to dispense with the abstract boundary entirely and trace through the centers of a domain's peripheral pixels<sup>20</sup>. This works well for domains that are only one pixel wide. However, it still does not eliminate double traversal for domains wider than one pixel and some form of backtracking is needed to deal with domains with shapes like  or .

### 6.1.2 Raster Neighborhood Coding

The most common use of raster neighborhood coding is to classify pixels on black/white images such as digital facsimile transmissions. A *neighborhood template* gives a context dependent probability estimation of the black/white value of a pixel. If the context dependent probabilities of occurrence of black and white pixels are unequal, the estimate reduces the information necessary to determine the classification of a pixel. The degree to which the probabilities of occurrence of the various symbols in an alphabet differ is called the *probability skew* of the alphabet.

The base information needed to code a two symbol alphabet is one bit per pixel. Any symbol probability skew in the context dependent probability estimator reduces this value. Clearly, once all pixels have been classified, contour information is readily available. An International Standards Organization (ISO) standard called the JBIG (Joint Bilevel Image Group) bilevel image compression algorithm is perhaps the best known implementation of this technique.

On the general image, at least four classifications, or colors, are needed to unambiguously delineate domain boundaries. Although the four-coloring problem (coloring all domains of a partition with only four colors and assigning different colors to all pairs of adjacent domains) is difficult, if one was available the raster neighborhood coder could be extended directly by doubling the number of possible pixel classifications.

One can dispense with the four-coloring and extend the raster neighbor coder with the simple expedient of *assigned edges*<sup>21</sup>. If pixels are thought of as square two dimensional lattice sites, there is a separator site to each pixel's north, south, east, and west. A separator site may or may not be *occupied*. Each pixel is assigned two of the four *separator sites* that surround it. Since for an image with  $N$  pixels there are  $2N$  possible separators, this assignment is sufficient to cover all possible partitions.

Typically, for a raster scan, each pixel is assigned the separator sites to its south and east. If a separator site is empty (no separator), then the two pixels on either side of the site are in the same domain. If it is full (a separator exists), they are not.

In addition, the probability estimation context is expanded to hold probability estimates of four symbols for each neighboring pixel, indicating the presence of zero, either, or both separators. For a four symbol alphabet, the base coding rate is two bits/pixel. Again, any symbol probability skew in the context dependent probability estimator reduces this value.

One disadvantage of the raster technique is that a decision must be made for every pixel site in an image even when the number of boundary separators is fairly sparse. The main advantage of the method is that it is inherently single pass and has highly local memory accesses.

## 6.2 A Boundary Partition Classification

We use the domain count,  $|P|$ , of a boundary partition and the pixel count,  $N$ , of its underlying image to develop a *boundary partition classification*. If  $|P| > \frac{N}{\log_2 N}$ , then a partition is *dense*. If  $|P| < \sqrt{N}$ , then a partition is *sparse*.

Recall from Chapter 1 that a piecewise smooth image partition cannot be dense. Further, when coding an image via a piecewise smooth approximation at the lowest bit rates the partition will be sparse or nearly so. Clearly, to code a piecewise smooth image model at the lowest possible bit rate, the boundary code must be optimized for sparse partitions.

For sparse partitions the chain code has an intuitive appeal as the lowest cost approach to boundary representation - if the problem of chain termination can be addressed. The stroke code of the next section does just that.

### 6.3 Chain Coding via Strokes

The stroke code develops a partition via a series of *strokes*. Each stroke consists of a start point, and one or two boundary chains. Each chain of a stroke is terminated upon encountering a previously decoded stroke or the image boundary. The key idea is that *corner encounters* with previously decoded strokes may or may not terminate a stroke. Further information is supplied by the encoder to disambiguate such encounters.

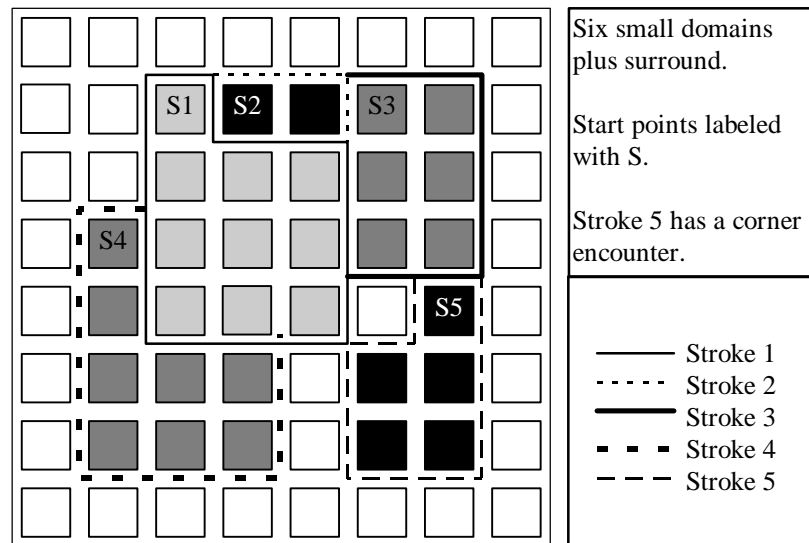


Figure 6.1  
Stroke Example

Strokes are decoded in turn: start, chain and termination information is interleaved in the code stream. Once the boundary is fully specified via strokes, domains may be grown by recursively joining groups of pixels that do not have a separator between them. Figure 6.1 has six domains of 1, 2, 5, 6, 8 and 9 pixels plus the surround. As can be seen, only five strokes are necessary to completely specify the boundary between these domains. After first covering some preliminaries, we develop stroke start points, move to termination disambiguation and then apply context dependent probability estimation to stroke chains.

### 6.3.1 Binary Decisions

The entire stroke code is comprised of binary decisions. Each decision is a yes or no answer to a question posed by the decoder. The information content of each question depends upon the uncertainty of its answer. Shannon<sup>22</sup> developed a quantitative measure of the average information in a series of binary decisions. If  $p_l$  is the probability of the least probable outcome and  $p_m$  is the probability of the most probable outcome in a sequence of decisions, then the *entropy* of the decisions is:

$$H = -p_l \log_2 p_l - p_m \log_2 p_m.$$

If the probabilities of occurrence of each symbol are fixed, the total *information* content of a binary code is  $d \cdot H$  where  $d$  is the total number of binary decisions in the code. We use a technique called the Laplacian estimator that dynamically develops estimated decision probabilities from the probability of their occurrence thus far in the code. If after  $n-1$  decisions of a series of decisions  $p_m(n-1)$  is the estimated probability of the occurrence of the most probable outcome, then the information content of a most probable outcome occurring at decision  $n$  is  $-\log_2 p_m$ . Similarly, the information content of a least probable outcome occurring at decision  $n$  is  $-\log_2 p_l$  or equivalently  $-\log_2(1-p_m)$ . The total information in a series of dynamically estimated decisions is the sum of the information of each outcome.



If  $n_m$  is the number of most probable outcomes and  $n_l$  is the number of least probable outcomes thus far in a series of  $n$  binary decisions, then  $p_m = \frac{n_m}{n_m + n_l}$  and  $p_l = \frac{n_l}{n_l + n_m}$  are the probability estimations for the next outcome. The sums  $n_m$  and  $n_l$  are called the *context* of a series of decisions. A sequence of binary decisions can have a number of contexts. For example, one of the contexts of the stroke code is the *stroke location* context. For all the pixels in an image, the decoder asks “Does a stroke start at this pixel?”. The number of yes answers and no answers are the stroke location context. Actually, there are several stroke location contexts, which we discuss in the next section.

Absent *prior* evidence to the contrary, all decision context sums are initialized with one most probable and one least probable outcome. This results in equal outcome probabilities until at least one decision is made.

### 6.3.2 Stroke Start Points

Each pixel in the image is a possible stroke location. The decoder scans the image in raster order and determines whether or not a pixel is a stroke location. The information associated with this process is called the *stroke location information*. Once a decision has been made that a pixel is a stroke location, different processing occurs depending upon the known boundary state at the stroke location.

Every stroke location can have a separator along either of its northern or western edges or both. If a stroke location already has one of its separators known from a previously decoded stroke, it has only one stroke chain that starts in the direction of the undecided edge. If it already has both of its separators known, it is not a stroke location and no deciding information is needed from the encoder.

If a stroke location has neither separator known, it can be either *bare* or *southeast corner connected*. A bare location has no edges impinging on its northwest corner. A southeast corner connected location has boundary separators along both the southern and eastern frontiers of the pixel to its northwest.

A bare location must be both northernmost and westernmost since it has no other edges with which to connect. The chain starting along its western edge and heading south is decoded first. If that chain is not *closed*, a second chain is decoded starting along the stroke location's northern edge and heading east. A closed chain returns to its start location heading in the opposite direction from its outset.

Name	Determining information
<b><i>ze</i></b>	Zero previously decoded edges, bare
<b><i>se</i></b>	Zero previously decoded edges, southeast corner to northwest
<b><i>oe</i></b>	One previously decoded edge
<b><i>te</i></b>	Two previously decoded edges

Table 6.1  
Stroke Location Contexts

There are four stroke location contexts corresponding to either one or two previously decoded edges, and bare and southeast corner connected locations where no previously decoded edges are known. Table 6.1 summarizes these contexts and gives them names. There is no information associated with the ***te*** context; these pixels are not possible stroke locations. The skew of the ***ze*** context is typically much higher than that of the ***se*** or ***oe*** contexts, thereby reducing the information of the three contexts relative to that of a single context containing all ***se***, ***oe***, and ***ze*** locations. On Figure 6.1 strokes one and four have ***ze*** start points. Strokes two, three and five have ***oe*** start points.

Since it can connect to the boundary corner to its northwest, a southeast corner connected location cannot know if it has a chain starting along its western edge, its northern edge or both. Further *chain start disambiguation information* is supplied by the encoder to differentiate between the three possibilities. Disambiguation takes two binary decisions. The first decision is both chains or one. If one, the second decision is north or west.

The ***se*** context also holds the statistics of the start disambiguation decisions. Three sums are maintained, the number of southeast corner connected locations with two stroke chains, the number with a northern chain, and the number with a western chain.



Figure 6.2  
Boundary and Stroke Start Points  
100 Domain Lena

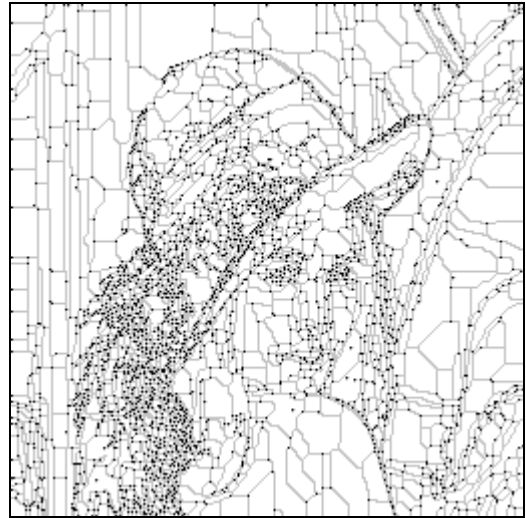


Figure 6.3  
Boundary and Stroke Start Points  
3200 Domain Lena

Figure 6.2 and Figure 6.3 show stroke locations as darker dots for 100 and 3200 domain Lena partitions. Note how larger domains may have many stroke locations along their periphery. Once a large domain is decoded, its periphery gives likely stroke locations. This is the source of the skew disparity between the *ze* and *se* or *oe* contexts.

### 6.3.3 Stroke Chain Termination

A stroke chain implicitly terminates if it makes a  $\perp$  intersection with the image boundary or with a previously decoded stroke. A corner intersection may or may not terminate a stroke chain. *Corner disambiguation information* is supplied by the encoder to distinguish the two cases. The corner disambiguation context, designated *cd*, holds the statistics of the disambiguation outcomes. Since stroke chains can only be terminated by intersection with another stroke, there are no *hanging* (unterminated) chains present at any point in the decode process. On Figure 6.1 stroke five has a continuing corner junction with stroke one and a terminating corner junction with stroke three.

### 6.3.4 Stroke Chains

Stroke chains cannot backtrack and are terminated by intersection with another stroke. Therefore, stroke chains are three direction chains: left, right straight. This three-way decision is reduced to two binary decisions via a two level coding tree. The first decision differentiates between straight and turn. The second between left and right.

The sixteen probability estimation contexts of Table 6.2 are used to reduce the information content of the stroke chains. Having multiple contexts allows us to increase the average skew of the binary decisions over that of a single context. These contexts are designed to capture the statistics of chains that result from the smoothing procedure of Chapter 0. A simple state machine determines the context used to code each chain event. Several contexts end in  $ss^+$  and their purpose is to differentiate straight sections that are known to be *long* (two or more straight events) from those that are not yet known to be long.

$ls$	previous event straight, left prior to that
$rs$	previous event straight, right prior to that
$ll$	previous event left, left prior to that
$rr$	previous event right, right prior to that
$lss^+$	at least two previous events straight, last turn left
$rss^+$	at least two previous events straight, last turn right
$rl$	previous event left, right prior to that
$lr$	previous event right, left prior to that
$rl(rl)^+$	$rl$ repeated more than once
$lr(lr)^+$	$lr$ repeated more than once
$l(lr)^+l$	$ll$ with at least one included $rl$
$r(lr)^+r$	$rr$ with at least one included $lr$
$(rl)^+s$	special case of $ls$
$(lr)^+s$	special case of $rs$
$(rl)^+ss^+$	special case of $lss$
$(lr)^+ss^+$	special case of $rss$

Table 6.2  
Stroke Chain Contexts

The main point is that these contexts are not simply the last few chain directions. For example, the  $lss^+$  context is *active* if the last two directions traveled were straight and the last turn previous to the straight section was a left turn. The  $lr(lr)^+$  context is active if the last direction was a right turn and all directions prior to that right were alternating turns beginning with a left turn. Taken together these contexts are designed to capture the behavior of the raster drawn boundary preferred by Chapter 0's smoothing procedure.

## 6.4 Experiments

Table 6.3 and Table 6.4 show the results of simulating the stroke code against unsmoothed and smoothed Lena partitions of between 100 and 3200 domains\*. The first column is the number of domains in the partition, the second column is the total number of separators in the boundary, and the third is the total number of strokes necessary to code the boundary. The fourth through seventh columns are the location, corner disambiguation, chain and total information for the stroke code. The location information column includes chain start disambiguation information. Since the information content of a sequence of binary decisions is a very good estimate of the number of bits necessary to code that sequence using an arithmetic coder, column seven is a good estimate of the total bit count necessary to code the partition.

---

\* Refer to Figure 5.15, Figure 5.16, Figure 6.2, and Figure 6.3 for visual representations of several test data sets.

Counts			Information			
Domains	Separators	Strokes	Location	Corner	Chain	Total
100	7929	96	948	33	9105	10085
200	10269	190	1712	59	12033	13804
400	15912	392	3219	99	20234	23552
800	19139	765	5638	245	24007	29890
1600	25966	1529	9850	419	34342	44611
3200	33601	2905	16647	1328	44153	62129

Table 6.3  
Stroke Data (Unsmoothed Lena)

Counts			Information			
Domains	Separators	Strokes	Location	Corner	Chain	Total
100	6151	99	949	5	4957	5911
200	7921	199	1656	13	6495	8165
400	12285	397	3073	35	11795	14903
800	15621	772	5367	149	15697	21213
1600	19331	1543	9007	313	20184	29503
3200	25396	2914	15065	1214	27752	44031

Table 6.4  
Stroke Data (Smoothed Lena)

The first thing to note is the significant difference between the code's performance on the smoothed and unsmoothed examples. This is due primarily to decreased chain direction entropy for the smoothed boundary. Direction entropy for the 100 domain unsmoothed Lena is 1.15, but for the corresponding smoothed example is only 0.8. Also of interest, is that corner disambiguation information is never a significant percentage of the total code length. This is really the main reason for the overall success of the code.

The last point to glean from Table 6.3 and Table 6.4 is that chain direction entropy increases and chain location information becomes more significant as the domain density increases. Even so, the results are good for the smoothed boundary right up to the dense threshold. This is especially significant since the code is designed to operate most efficiently on sparse partitions.

Since the stroke code handles the chain termination problem with very little overhead, it appears to be superior to any previously reported chain code for partitioned general images. We now examine how it compares with raster neighborhood codes. As a lower bound for raster codes, we can look at the performance of the Q-Coder<sup>23</sup> on *northwest-black* boundary images. A northwest-black boundary image is formed by turning a pixel black if it is adjacent to the northern or western frontiers of a domain and white otherwise. This results in a black/white image similar to those of Figure 6.2 and Figure 6.3\*.

Counts		Information		
Domains	NW Pixels	Q-Code	Separators	Stroke Code
100	6647	12643	7929	10085
200	8641	16343	10269	13804
400	13086	25407	15912	23552
800	15751	30087	19139	29890
1600	21108	38014	25966	44611
3200	26792	43607	33601	62129

Table 6.5  
Stroke Code vs Q-Code  
Unsmoothed Lena

This means that we are comparing a one bit/pixel base rate raster code to the stroke code. Remember, a one bit/pixel base rate raster code can only yield complete boundary information for black/white images. A general rule of thumb is that a one bit raster code only knows about as many separators as there are northwest-black pixels. The greater the difference between the number of separators and northwest-black pixels, the more information is needed by the one bit raster code to completely specify the boundaries of the equivalent partitioned general image. Clearly the Q-Coder used in this fashion produces shorter code streams than any two bit/pixel base rate raster neighborhood code.

---

\* Just turn the gray pixels black.

Counts		Information		
Domains	NW Pixels	Q-Code	Separators	Stroke Code
100	5148	7999	6151	5911
200	6620	10137	7921	8165
400	10118	16609	12285	14903
800	12942	20746	15621	21213
1600	15998	24349	19331	29503
3200	20473	30132	25396	44031

Table 6.6  
Stroke Code vs Q-Code  
Smoothed Lena

Table 6.5 and Table 6.6 compare the stroke code and northwest-black Q-Code for the previously examined unsmoothed and smoothed Lena partitions. The first column is the number of domains in the partition, the second column is the number of northwest-black pixels in the Q-coded test image, the third column is the corresponding Q-code length. The fourth column is the number of separators in the partition. Note how the number of separators becomes increasingly greater than the number of northwest-black pixels as the number of domains increases. The last column is the stroke code length. Surprisingly, the stroke code is superior to the Q-code all the way up to 800 domains.



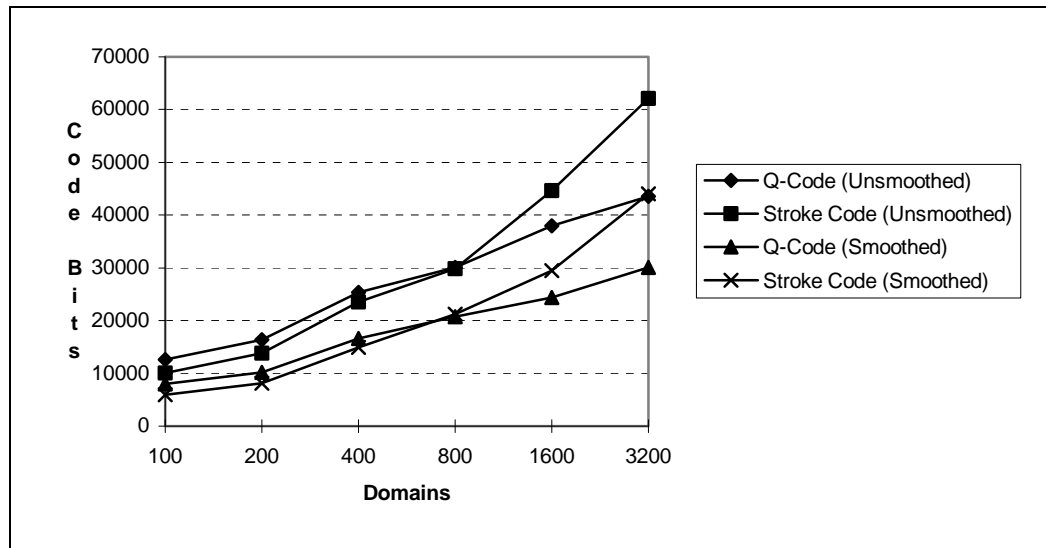


Figure 6.4  
Stroke Code vs Q-Code  
Smoothed and Unsmoothed Lena

Figure 6.4 charts the data of Table 6.5 and Table 6.6. Again note the crossover point at around 800 domains for both the smoothed and unsmoothed groups of partitions. It is evident that the stroke code is superior to any two bit raster neighborhood code for sparse partitions. What is not yet known is exactly where the cross-over point with two bit raster codes lies or even if it exists.

## 6.5 Summary

We classified image partitions as sparse, dense or between sparse and dense. We developed the stroke code for representing the boundaries of a partitioned image. It is the first code to handle the three direction chain termination problem. We apply the raster drawn boundary criteria of Chapter 5 to develop improved chain direction probability estimation contexts. We showed the stroke code to be superior to raster neighbor codes on sparse image partitions.

## 7. Geometry Implicit Coding of Two Dimensional Polynomials

We continue with our task of minimally representing a piecewise-smooth image model by developing methods to code each domain's polynomial intensity function. We assume that the domain boundaries of the image model are coded separately and are available to the polynomial coder and decoder. We develop the method of *sentinel points*, whereby certain points in a domain, known only from the domain's geometry, are used so solve a system of simultaneous equations that yield the domain's intensity function. With this technique, the locations of the sentinel points are known implicitly and only their values must be decoded to recover the model.

A global parameter associated with each image model is the maximum order of its domains' polynomial intensity functions. The polynomial order of a domain is reduced when it does not have sufficient support for all the terms of the model's maximum order polynomial. The stability parameter of the modified Cholesky method of Chapter 3 is the arbiter of what polynomial terms are supported by a domain. This stability parameter is another global parameter of each model.

The intensity functions of a model's domains can be represented with varying accuracy. This is accomplished via variable quantization of the sentinel point values. The amount of quantization applied to sentinel point values is determined implicitly by the size of the associated domain. Three global parameters determine the maximum and minimum quantization applied to sentinel point values and the distribution of the sentinel point values into the various quantizers.

### 7.1 Background

Typically<sup>11,24</sup>, segmentation-based image coding schemes allocate eight bits per polynomial coefficient when estimating bit rates. Kunt<sup>24</sup> suggests recovery of polynomial coefficients from "regularly spaced" eight bit data values. In Section 7.3, we develop an algorithm for determining data value locations for first through third order approximating polynomials.

These locations are constrained to lie within their associated domain and as such, are not regularly spaced.

## 7.2 Sentinel Points and Polynomial Reconstruction

Given a connected discrete two dimensional domain,  $\Delta$ , we would like to find the  $(x, y)$  coordinates of  $n$  *sentinel points* that can be used to optimally recover any  $n$  term polynomial,  $z = f(x, y)$ , over the domain. All points of  $\Delta$  have positive coordinates and the range of  $z$ ,  $R_z$ , is limited to lie between two positive integers  $z_L$  and  $z_H$ .

If given  $n$  tuples  $(x_i, y_i, z_i)$  we can construct an  $n$  term polynomial through them by solving a system of simultaneous equations. For example, given the two dimensional linear polynomial:

$$z = ix + jy + k,$$

and three points on its surface,  $(x_1, y_1, z_1)$ ,  $(x_2, y_2, z_2)$ , and  $(x_3, y_3, z_3)$ , we can solve the system:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

to yield the coefficients  $i$ ,  $j$ , and  $k$ .

The accuracy with which we can recover the polynomial coefficients using such a system is limited by the accuracy to which the  $z_i$  are known. If the  $z_i$  are known to a fixed precision, our problem is to find the sentinel points,  $(x_i, y_i)$ , that no matter what the polynomial yields its best approximation given the  $z_i$ .

One may be tempted to specify a regular pattern of sentinel points by inscribing  $\Delta$  inside of a figure such as a rectangle. Doing this, however, does not constrain the values of the sentinel

points to  $R_z$ . For higher order polynomials, points even slightly outside of  $\Delta$  can take on values significantly beyond  $R_z$ . In other words,  $\Delta$ 's polynomial function is guaranteed to be within its range only over  $\Delta$  and deviations outside of  $\Delta$  are not bounded. Another way of saying this is that the polynomial is interpolating over the domain and extrapolating outside of the domain.

Since our goal is to code a polynomial as accurately and using as few bits as possible, we can take advantage of the known  $R_z$  to maximize the precision to which the  $z_i$  are known if we constrain the sentinel points to lie within their corresponding domain. This constraint leads to the sentinel point selection algorithm of the next section.

### 7.3 Choosing Sentinel Points

We now present algorithms for choosing optimal sentinel point locations for four different polynomials:

- $z = k$
- $z = ix + jy + k$
- $z = fx^2 + gxy + hy^2 + ix + jy + k$
- $z = ax^3 + bx^2y + cxy^2 + dy^3 + fx^2 + gxy + hy^2 + ix + jy + k$ .

These points are optimal in the sense that their locations minimize the derivative of the recovered polynomial coefficients with respect to changes in their data values

#### 7.3.1 Zero Order System

For the zero order system, the solution is trivially any point in  $\Delta$  since all points have the same value. We pick the point whose  $y$  coordinate is less than or equal to any other point in  $\Delta$  and whose  $x$  coordinate is less than any other point with the same  $y$  coordinate. Since our

coordinate system has  $x$  increasing to the right and  $y$  increasing downward, this point is designated the northwestern most point in  $\Delta$ .

### 7.3.2 First Order System

The two-dimensional linear polynomial is:

$$z = ix + jy + k.$$

We desire to find three points,  $(x_1, y_1)$ ,  $(x_2, y_2)$ , and  $(x_3, y_3)$ , that when used to solve the system:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

yield the best values of  $i$ ,  $j$ , and  $k$  given that there may be error in the values,  $z_1$ ,  $z_2$ , and  $z_3$ .

The determinant of the *sentinel matrix* is:

$$D = (x_1 \cdot y_2 - x_1 \cdot y_3 - x_2 \cdot y_1 + x_2 \cdot y_3 + x_3 \cdot y_1 - x_3 \cdot y_2)$$

and the solution for the polynomial coefficients is:

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = \frac{1}{D} \begin{bmatrix} y_2 - y_3 & -y_1 + y_3 & y_1 - y_2 \\ -x_2 + x_3 & x_1 - x_3 & -x_1 + x_2 \\ x_2 \cdot y_3 - x_3 \cdot y_2 & -x_1 \cdot y_3 + x_3 \cdot y_1 & x_1 \cdot y_2 - x_2 \cdot y_1 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}.$$

If we differentiate  $i$ ,  $j$ , and  $k$  with respect to  $z_1$ ,  $z_2$ , and  $z_3$  we can find the sensitivity of the coefficients with respect to changes in sentinel point values. If we jointly minimize the coefficient sensitivities, we can obtain the sentinel points that give the best possible accuracy for  $i$ ,  $j$ , and  $k$  given an error bound on their values. Differentiating with respect to  $z_1$  yields:

$$\frac{d}{dz_1} \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{bmatrix} \frac{-y_2 + y_3}{D} \\ \frac{x_2 - x_3}{D} \\ \frac{x_3 \cdot y_2 - y_3 \cdot x_2}{D} \end{bmatrix}.$$

Differentiation with respect to all three sentinel point values yields nine sensitivities, three for each coefficient. Of course it is not possible to bring all sensitivities to zero. We could attempt a mean squares solution, but we really desire an approximately accurate solution that can be quickly calculated.

One observation is that all sensitivities are divided by  $D$ . If  $|D|$  is large, all sensitivities are small. It is not immediately obvious how to make  $|D|$  large, but if we add the further constraint that  $y_1 = y_2$ , then  $D$  reduces to:

$$D_{\text{reduced}} = (-x_1 + x_2) \cdot (-y_1 + y_3).$$

What we want is two points with the same  $y$  coordinate that maximally differ in  $x$  and a third point that maximally differs in  $y$  from the other two. Or if we add the constraint  $x_1 = x_2$ , we want two points with the same  $x$  coordinate that maximally differ in  $y$  and a third point that maximally differs in  $x$  from the other two. The coordinate equivalency constraint allows us to *separate* the problem into one in  $x$  and one in  $y$ . An optimal solution to even the separated problem requires  $O(m \cdot n)$  time.  $m$  is  $\Delta$ 's extent in  $x$  and  $n$  is its extent in  $y$ .

To reduce the algorithmic complexity, we can adopt a greedy approach and maximize each term of the reduced  $D$  in turn:

1. First choose  $\Delta$ 's maximal length rectilinear chord and make its endpoints  $(x_1, y_1)$  and  $(x_2, y_2)$ . Designate this chord the domain's *primary chord*.
2. Find the point on  $\Delta$ 's periphery that is maximally distant in the opposing coordinate from the primary chord and make that point  $(x_3, y_3)$ . Designate the

line drawn perpendicularly from the primary chord and through  $(x_3, y_3)$  the secondary chord.

3. Break ties by choosing chords at minimal perpendicular distance from the centroid of  $\Delta$ .

Since the points on the periphery of  $\Delta$  are examined at most twice, the *greedy algorithm for choosing linear sentinel points* is  $O(m + n)$ .

### 7.3.3 Second Order System

Solving for the sentinel points for the second order polynomial:

$$z = fx^2 + gxy + hy^2 + ix + jy + k$$

is intractable in its intact form. To make things tenable, we again separate the problem. Using a greedy approach, we solve first for sentinel points that yield  $x$  coefficients, next for points that yield  $y$  coefficients and then a final point for the mixed coefficient.

Using the method of the previous section, we develop a system of equations in three unknowns to find the best three points for a single dimensional domain. Separating out the  $x$  problem yields the following sentinel matrix:

$$\begin{bmatrix} x_0^2 & x_0 & 1 \\ x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \end{bmatrix},$$

whose determinant is:

$$D = (x_0 - x_1) \cdot (-x_0 + x_2) \cdot (-x_1 + x_2).$$

### 7.3.3.1 Primary Chord

If the points are arbitrarily ordered,  $x_0 \leq x_1 \leq x_2$ , we can maximize the middle factor by choosing the endpoints of our one dimensional domain,  $\mathbf{X}$ , for  $x_0$  and  $x_2$ . Differentiating  $D$  with respect to  $x_1$ :

$$\frac{d}{dx_1} D = (x_0 - x_2) \cdot (-x_1 + x_2) - (x_0 - x_1) \cdot (-x_0 + x_2)$$

and finding a local minimum yields:

$$x_1 = \frac{x_2 + x_0}{2},$$

which is simply the midpoint of  $\mathbf{X}$ .

Now that we know how to maximize  $|D|$  for a given  $\mathbf{X}$ , we can find the maximal  $|D|$  rectilinear chord of our two dimensional domain. Noting the equivalency of interchanging  $x$  and  $y$ , this *primary* chord can be either horizontal or vertical. We drop the  $x$  notation and designate the three sentinel points of the primary chord  $p_1$ ,  $p_2$ , and  $p_3$ .

### 7.3.3.2 Secondary Chord

We next find a *secondary* chord in the opposing dimension that has *maximal*  $|D|$  *relative to the primary chord*. By relative, we mean that only two of the  $D$  points,  $s_1$ ,  $s_2$ , and  $s_3$  of a secondary chord are *independent* of the primary chord. The *dependent*  $D$  point of a secondary chord is its intersection with the primary chord.

If  $s_p$  is a secondary chord's dependent  $D$  point, the *relative*  $D$  of a secondary chord,  $D_s$ , is found by fixing one its  $D$  points, say  $s_1$ , at  $s_p$ . To maximize  $D_s$  our task is reduced to finding optimal locations on the chord for  $s_2$ , and  $s_3$ . Clearly, the point on a secondary chord furthest from the intersection point is one of its maximal  $|D_s|$  points.



To find the final independent maximal  $|D_s|$  point of a secondary chord, one option would be an exhaustive search among the points of the chord. To keep the secondary chord algorithm  $O(1)$ , however, we use the  $\mathbf{X}$  algorithm to find a secondary chord's three best independent points. We then substitute  $s_p$  for each of these points and keep the result with maximal  $D$ . This takes only three interchanges and comparisons.

Next we find the *chord combination* that maximizes the product  $|D_p \cdot D_s|$ . Since we have already fixed the primary chord, this search consists of finding the secondary chord with maximal  $|D_s|$ . The two independent maximal  $|D_s|$  points of the secondary chord together with the three points of the primary chord give us five of the desired six sentinel points for the second order polynomial. These points define the unmixed polynomial terms.

### 7.3.3.3 Mixed Term Sentinel Points

To find the mixed term sentinel points, we recognize that the primary and secondary chords define a coordinate system with its origin at the chord intersection. We then maximize  $D_m$  of the mixed term sentinel matrix in the *chord coordinate system*. The mixed term sentinel matrix of  $\Delta$  contains only a single component:  $[x_1 y_1]$ . The point in  $\Delta$  that maximizes this product in the new coordinate system is the mixed term sentinel point. That point can be found by procedurally searching through all points of the periphery of  $\Delta$ .

### 7.3.3.4 Computational Complexity

- Finding the mixed term sentinel point is proportional to the length of perimeter of  $\Delta$  or  $O(m+n)$ .
- Finding the primary chord is  $O(m+n)$ : we examine three points on each rectilinear chord of a domain.
- Finding the secondary chord is  $O(n)$  or  $O(m)$ .
- The overall computational complexity is  $O(m+n)$ .

Up to this point we have assumed that all one dimensional chords,  $\mathbf{X}$ , of  $\Delta$  are connected; this may not be the case. Accordingly, the midpoint of  $\mathbf{X}$  may not lie in  $\mathbf{X}$ . If so, we modify

the  $\mathbf{X}$  algorithm to substitute the point lying in  $\mathbf{X}$  that is closest to its midpoint for  $x_1$ . Doing this increases the sentinel point computational bound to  $O(m \cdot n)$ . Fortunately, domains that press against this bound are rare in our image coding application.

### 7.3.3.5 Summary

To summarize, the second order sentinel point algorithm is:

1. Use the  $\mathbf{X}$  algorithm to find the rectilinear chord of  $\Delta$  with maximal  $|D|$  and obtain three sentinel points.
2. Using the secondary chord algorithm, find  $\mathbf{X}_s$  that maximizes  $|D_s|$  and obtain two additional sentinel points.
3. Find the point on the perimeter of  $\Delta$  that maximizes  $|D_m|$  and use it as the final sentinel point.

### 7.3.4 Third Order System

To solve the sentinel point problem for the third order polynomial

$$z = ax^3 + bx^2y + cxy^2 + dy^3 + fx^2 + gxy + hy^2 + ix + jy + k$$

we again separate the problem.

#### 7.3.4.1 Third Order $\mathbf{X}$ Interior Points

The separated third order sentinel matrix in  $x$  is:

$$\begin{bmatrix} x_0^3 & x_0^2 & x_0 & 1 \\ x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ x_3^3 & x_3^2 & x_3 & 1 \end{bmatrix}.$$

The maximal  $D$  interior points for the third order  $\mathbf{X}$  problem are:

$$x_1 = \begin{bmatrix} \frac{x_0 + x_3}{2} + \frac{\sqrt{5}}{10} \cdot (x_3 - x_0) \\ \frac{x_0 + x_3}{2} - \frac{\sqrt{5}}{10} \cdot (x_3 - x_0) \end{bmatrix}$$

### 7.3.4.2 Mixed Coefficients

The mixed coefficient sentinel system now contains more than one element:

$$\begin{bmatrix} x_1 \cdot y_1 & x_1^2 \cdot y_1 & x_1 \cdot y_1^2 \\ x_2 \cdot y_2 & x_2^2 \cdot y_2 & x_2 \cdot y_2^2 \\ x_3 \cdot y_3 & x_3^2 \cdot y_3 & x_3 \cdot y_3^2 \end{bmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

$D$  for the mixed term matrix is:

$$D = x_2 \cdot x_1 \cdot y_2 \cdot x_3 \cdot y_3 \cdot y_1 \cdot (y_2 \cdot x_1 - x_3 \cdot y_2 - y_3 \cdot x_1 + x_3 \cdot y_1 - y_1 \cdot x_2 + x_2 \cdot y_3)$$

Working in the chord coordinate system we greedily maximize  $|D_m|$  as follows. First temporarily set  $(x_2, y_2)$ , and  $(x_3, y_3)$  to zero and find the sentinel point that maximizes the remainder of  $|D_m|$ :

$$x_1 \cdot y_1$$

Next reinstate  $(x_2, y_2)$  and find the point that maximizes

$$x_2 \cdot y_2 (x_1 \cdot y_2 - x_2 \cdot y_1)$$

Finally, find the point that maximizes the overall  $|D_m|$ :

$$x_3 \cdot y_3 (-y_2 \cdot x_1 + x_3 \cdot y_2 + y_3 \cdot x_1 - x_3 \cdot y_1 + y_1 \cdot x_2 - x_2 \cdot y_3)$$

Since none of the mixed terms are factors of one another, we can constrain the mixed term search to the periphery of  $\Delta$ , and the algorithmic complexity bounds developed for the second order problem still apply. The third order sentinel algorithm is  $O(m+n)$

### 7.3.5 Examples

We end this section with some sentinel point examples. Figure 7.1 shows the zero order sentinel points for synthetic image Syn15. They are simply northwestern most. Figure 7.2 shows the first order sentinel points for the same image. Note that all points are peripheral.

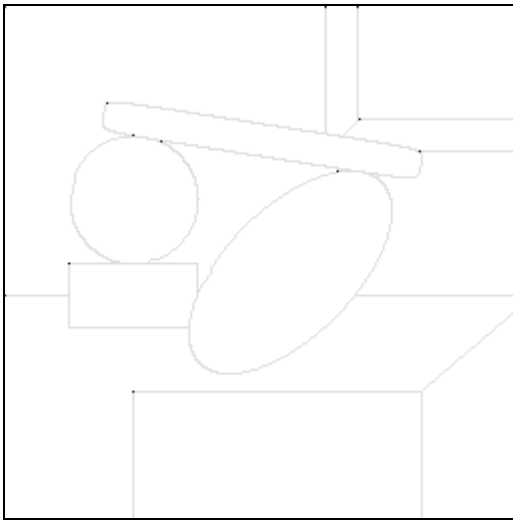


Figure 7.1  
Zero Order Sentinel Points for Syn15

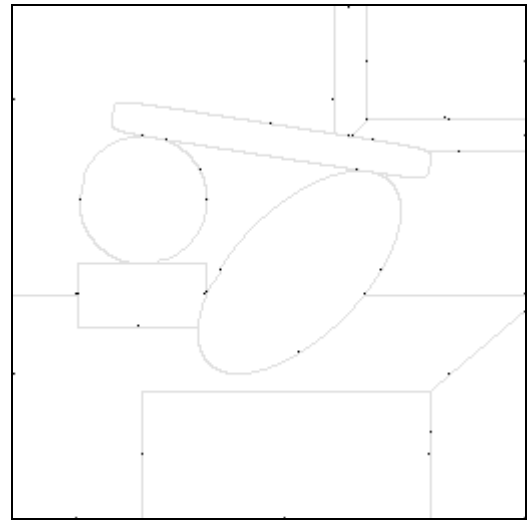


Figure 7.2  
First Order Sentinel Points for Syn15

Figure 7.3 shows the second order sentinel points for Syn15. Interior points are now developing. Figure 7.4 shows the third order sentinel points. Three points per domain are now interior.

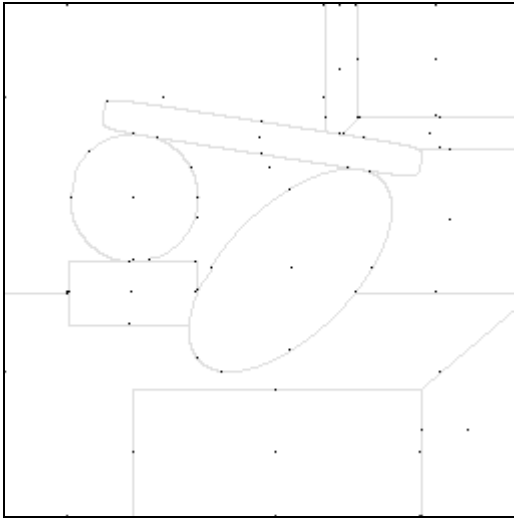


Figure 7.3  
Second Order Sentinel Points for Syn15

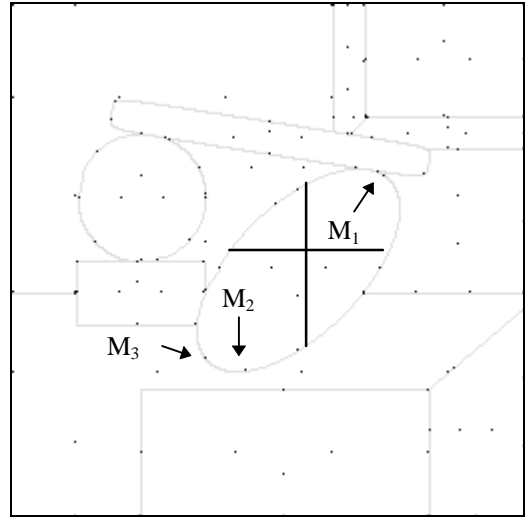


Figure 7.4  
Third Order Sentinel Points for Syn15

To help understand how the sentinel points fall, let's find them for the ellipse in Figure 7.4. First, since this is a third order example, there are four primary chord sentinel points. For this domain the primary chord is horizontal and its four sentinel points are found just below the dark horizontal line. The three secondary chord points align vertically to the left of the vertically drawn line. The three mixed coefficient points are labeled  $M_1$ ,  $M_2$ , and  $M_3$ .

As a final example, Figure 7.5 and Figure 7.6 show third order sentinel points for 100 and 800 domain partitions of Lena.

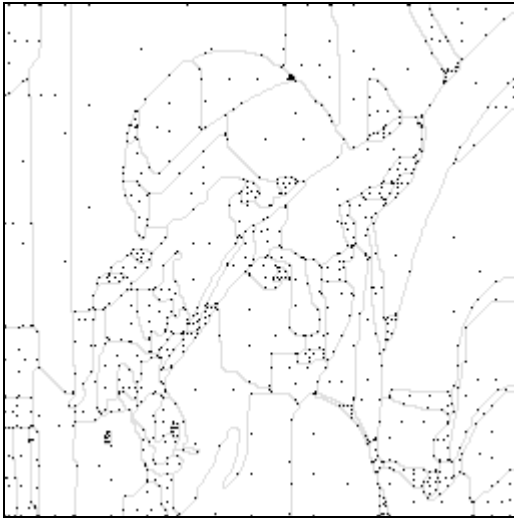


Figure 7.5  
Third Order Sentinel Points for a 100 Domain Lena



Figure 7.6  
Third Order Sentinel Points for an 800 Domain Lena

#### 7.4 Sentinel Points of Under-Constrained Domains

We saw in Chapter 3 that a domain may not necessarily support all the terms of a given polynomial order. Each image model has an associated maximum order of its domains' polynomial intensity functions. The polynomial order of a domain is reduced when it does not have sufficient support for all the terms of the model's maximum order polynomial. The stability parameter of the modified Cholesky method of Chapter 3 is the arbiter of what polynomial terms are supported by a domain

When using sentinel points in a polynomial coding procedure, the coder and decoder both know the image model's global Cholesky stability factor once it has been transmitted. Using the same Cholesky method with the same stability factor as the encoder, the decoder can know which sentinel points to expect without any additional information from the encoder beyond the domain's geometry.

The terms of support are a function of domain geometry only, and can be determined before sentinel point discovery commences. If we assign one sentinel point to each polynomial term, we need only develop sentinel points for the supported terms. For example, if the maximum global model order is three and the  $x^3$  polynomial term is not supported by a particular domain and is the only term in  $x$  not supported, we reduce the  $x$  term sentinel point search

to the second order  $\mathbf{X}$  problem. If both  $x^2$  and  $x^3$  are not supported by a domain, the  $x$  term sentinel point search reduces to the first order  $\mathbf{X}$  problem.

Unsupported mixed term sentinel points can be handled easily as well. For second order models, we simply eliminate the single mixed term sentinel point if it is not supported. For third order models, we simply bypass any unsupported terms in the greedy algorithm for locating mixed term sentinel points.

## 7.5 Quantization

Since we want to apply sentinel point polynomial reconstruction to lossy image coding, we now look at how the reconstruction degrades when the sentinel point values are quantized. Quantization of a set of data values constrains those values to take on a fixed number of values. Each value is a function of the quantizer step size,  $Q_{step}$ . The *quantization function* maps each data value to its corresponding quantized value. The quantization function that we use is:

$$P_Q = \frac{\text{round}(P)}{Q_{step}} \cdot Q_{step} + \frac{Q_{step}}{2}$$

where all operations except *round()* are integer operations.

After expending so much effort finding sentinel points, we would hope that quantization of the sentinel points of  $\mathbf{\Delta}$  would not increase the error of our approximating polynomial above some reasonable bound. We use the following definitions to develop such a bound. Define the MSE of our full precision approximation as  $E_F$ . Define the additional error introduced by *directly* quantizing the data values predicted by the full precision approximation as  $E_{QD}$ . Define the MSE of the polynomial recovered with quantized sentinel points as  $E_{QS}$ .

If we assume independent noise sources and uniform distribution of data values into quantizer buckets, the MSE of the polynomial extracted from the quantized sentinel points should be bounded as follows:

$$E_{QS} \leq E_F + E_{QD}.$$

The quantity

$$G = \frac{E_{QS} - E_F}{E_{QD}}$$

is a measure of how well this bound is met. Good sentinel point selection results in  $G$  values near one. Essentially,  $G$  is the ratio of the error in the data values predicted by the quantized sentinel points to the error in the data values introduced by direct quantization of the values predicted by the full precision approximation.

$Q_{step}$	$E_F$	$E_{QD}$	$E_F + E_{QD}$	$E_{QS}$	PSNR
0	126.8	0	126.8	126.8	27.10
1	126.8	.25	127.1	127.3	27.08
2	126.8	.5	127.3	127.6	27.07
4	126.8	1.5	128.3	128.6	27.04
8	126.8	5.5	132.3	134.2	26.85
16	126.8	21.5	148.3	149.5	26.39
32	126.8	85.5	212.3	224.4	24.62
64	126.8	341.5	468.3	448.6	21.61.
128	126.8	1365.5	1492.3	1228.3	17.23

Table 7.1  
Quantization Error  
100 Domain Lena

Table 7.1 shows the error introduced by quantizing the sentinel points of a 100 domain partition of Lena. The first column is the quantizer step size used to generate columns two and four. The second column is the MSE of the unquantized piecewise-smooth third order model of the image. The third column is the additional noise expected from direct quantization of the intensity values produced by the full precision model. The fourth column is the sum of columns two and three. The MSE of the model coded via quantized sentinel points is in the fifth column. Although  $E_{QS}$  is slightly greater than  $E_F + E_{QD}$ , the difference is quite small.



The worst value of  $G$  is 1.14, corresponding to a  $Q_{step}$  of 32. Interestingly,  $G$  becomes less than one for quantizer step sizes greater than 32.

$Q_{step}$	$E_F$	$E_{QD}$	$E_F + E_{QD}$	$E_{QS}$	PSNR
0	25.3	0	25.3	25.3	34.10
1	25.3	.25	25.6	26.4	33.92
2	25.3	.5	25.8	26.6	33.89
4	25.3	1.5	26.8	27.6	33.73
8	25.3	5.5	30.8	33.1	32.93
16	25.3	21.5	46.8	52.4	30.94
32	25.3	85.5	110.8	128.1	27.06
64	25.3	341.5	366.8	378.6	22.39
128	25.3	1365.5	1390.8	1247.8	17.17

Table 7.2  
Quantization Error  
800 Domain Lena

Table 7.2 shows the same data for an 800 domain Lena partition. Again the worst value of  $G$ , 1.20, occurs for a  $Q_{step}$  of 32. This phenomenon is currently inexplicable.

Figure 7.7 charts the square root of columns  $E_{QD}$  and  $E_{QS}$  of Table 7.1 and Table 7.2. The RMS model error increases essentially as  $E_{QS} = \sqrt{E_F + E_{QD}}$  which is as expected for two independent sources of error. Particularly of interest is that models with higher initial error can be quantized quite heavily before additional distortion is introduced.

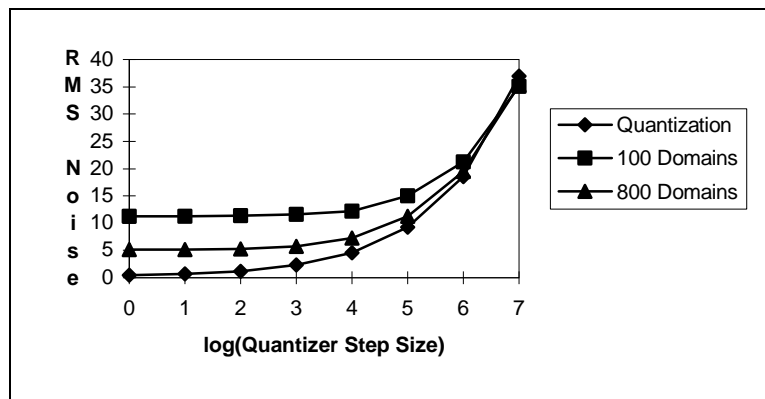


Figure 7.7  
Uniform Quantization Noise  
and Third Order Lena Examples

### 7.5.1 Variable Quantization

Studies of human perception<sup>4</sup> have shown that the eye is less sensitive to intensity variations of higher spatial frequency. We can take advantage of this knowledge when coding an image, and quantize smaller domains more heavily than larger domains. Smaller domains can also be quantized more heavily due to the nature of the extraction algorithms used in Chapters 3 and 4.

Since smaller domains have a larger periphery-to-area ratio, they are preferentially merged to minimize the boundary length term of the model cost function. It is clear that the smaller the domain, the more its intensity must differ from its neighbors for it to persist through the domain growing process. Since small domains differ significantly in intensity from their neighbors, they can be quantized more heavily before distortion becomes noticeable.

We introduce a simple mechanism for implicit quantization based upon domain size. For each image we specify three parameters.  $Q_l$  is the quantizer step size to use for the largest domains in the image.  $Q_h$  is the quantizer step size to use for the smallest domains in the image. The third parameter is a knee value,  $k$ , that spreads the sentinel points into the different quantizer step sizes in use.

Domains whose size is greater than  $k$  are quantized with a step size of  $Q_l$ . Domains whose size is in the range  $\frac{k}{2} \leq 2 < k$  are quantized with the next larger quantizer step size. At each factor of two reduction in size quantization becomes increasingly coarse until quantization reaches  $Q_h$  in which case all smaller domains are quantized at that step size. The only overhead associated with this implicit quantization scheme is the information necessary to transmit the three parameters,  $Q_l$ ,  $Q_h$ , and  $k$  to the decoder.

The *implicit size rule* combined with the size and fidelity dependencies allow for significant quantization over a wide range of compression ratios. When compression is high, the overall error is already significant before quantization and therefore quantization can be quite heavy before it introduces additional distortion. When compression is low, most of the domains are quite small and can be quantized more heavily.

Table 7.3 and Table 7.4 show the distribution of quantization step sizes for sentinel points of the previously discussed 100 and 800 domain Lena partitions. There are two quantizer step sizes in use in the 100 domain partition and five in the 800 domain partition. The knee parameter,  $k$ , is 200 and 50 respectively.

Quantizer Step Size	Coefficients
16	549
32	313

Table 7.3  
Quantization Distribution  
100 Domain Lena

Quantizer Step Size	Coefficients
2	1440
4	1062
8	1665
16	1467
32	1077

Table 7.4  
Quantization Distribution  
800 Domain Lena

Table 7.5 shows the corresponding error when quantizing the sentinel points as above. The increase in PSNR over the that of the unquantized model is less than one dB for both partitions.

Domains	MSE	RMS Error	PSNR
100	155.0	12.5	26.23
800	30.10	5.49	33.35

Table 7.5  
Lena with Variable Quantization

## 7.6 Optimizing Quantized Sentinel Point Values

The models extracted by the procedures of Chapters 3 and 4 do not account for polynomial coefficient quantization. Quantization introduces independent distortion that we can attempt to minimize by perturbing the values of sentinel points. Table 7.6 shows the results of performing a simple greedy optimization procedure on the sentinel points of two Lena and two Cameraman partitions. The PSNR is improved by an average of over .5 dB for each partition.

The sentinel value optimization algorithm works by examining each sentinel point in turn and perturbing it into the next higher and next lower quantizer buckets. Perturbations that lower the model's MSE are kept and those that do not are discarded.

The moment methods of Chapter 2 are used in this procedure. The polynomial coefficients are recalculated after each sentinel point perturbation using the Cholesky method of Chapter 2 on the moment set that contains only the sentinel points. These coefficients together with the natural and forcing moments of the full domain, are used in the error calculation. Using this technique, moment perturbations can be accomplished in  $O(1)$  time.

Image	domains	MSE	PSNR	Optimized MSE	Optimized PSNR
Lena	100	182.4	25.52	155.0	26.23
Lena	800	34.15	32.80	30.10	33.35
Cameraman	100	252.5	24.11	230.5	24.50
Cameraman	800	88.9	28.64	81.3	29.03

Table 7.6  
Optimizing Sentinel Point Values

## 7.7 Coding

We predictively code the quantized sentinel point values. Both the coded values and the prediction are specified in the *quantized data space*. In other words quantizer bucket differences

are transmitted and applied to a predicted quantizer bucket. Sentinel point difference values are transmitted for each domain in raster scan order

### 7.7.1 Quantization Model

In Section 7.4 we presented the quantization function used to apply quantization to sentinel point values:

$$P_Q = \frac{\text{round}(P)}{Q_{step}} \cdot Q_{step} + \frac{Q_{step}}{2}.$$

When coding these quantized values, we transmit only the *quantizer bucket* into which the data point falls. The quantizer bucket of a sentinel point,  $P$ , is developed using the *forward quantization function*:

$$Q_b = \frac{\text{round}(P)}{Q_{step}}.$$

The quantized sentinel point value is recovered from a sentinel point bucket using the *inverse quantization function*:

$$P_Q = Q_b \cdot Q_{step} + \frac{Q_{step}}{2}.$$

### 7.7.2 Prediction

The information necessary to specify the sentinel points of a domain is reduced via *prediction*. A predicted value for each sentinel point is developed and the *difference*,  $Q_d$ , between the prediction and the actual value is coded. The predictor used is the average value of the previously decoded points of the same domain. Using this simple *intra* predictor, the first point of each domain must be coded without prediction.

As each  $Q_d$  is received it is added to the quantizer bucket,  $Q_p$ , of the current prediction to calculate the quantizer bucket of the associated sentinel point,  $Q_b$ .  $Q_b$  is used to calculate  $P_Q$

via the inverse quantization function. Each inverse quantized sentinel point is added to the sentinel point list for its domain and is also used to update the predictor.

The predictor value is the running total of the  $P_Q$  values of the sentinel points received thus far for each domain.  $Q_p$  is calculated from this running sum as needed by dividing by the number of points in the sum and applying the forward quantization function.

Since sentinel point values can be variably quantized depending upon the size of their associated domain, there are several  $Q_{step}$  values in use for each image. There is one probability estimation context for each  $Q_{step}$  in use. The symbol probabilities of each  $Q_{step}$  context are used to reduce the entropy of the differences coded in that context. Unpredictable information falls into a *base* context.

The coding model for differentially coded sentinel point values is not yet complete. A two-pass, or  $n \log_2 n$ , entropy estimate is used to obtain the data of the following section and that of Sections 2.3 and 8.3. The two-pass entropy estimate is not attainable in practice and the sentinel point data should be viewed in that light. There are two practical problems with the two-pass entropy measure. First, symbol probability estimates are determined after all symbols have been counted. Obviously, this cannot be done by a real decoder. Second, possible symbols that do not occur anywhere in a simulated code stream do not contribute to the code string length. In any practical coder, a minimum probability must be assigned to every possible symbol regardless of occurrence in a particular simulated code stream. Any minimum probability assigned to symbols that do not appear increases the entropy of symbols that do appear in the code stream.

### 7.7.3 Experiments

Table 7.7 and Table 7.8 show sentinel point entropy for the 100 and 800 domain Lena examples.

Context	Sentinel Points	Entropy
base	962	3.12
16	549	3.16
32	313	2.90

Table 7.7  
Sentinel Point Entropy  
100 Domain Lena

Context	Sentinel Points	Entropy
base	7511	4.01
2	1440	5.02
4	1062	4.63
8	1665	3.90
16	1467	3.23
32	1077	2.63

Table 7.8  
Sentinel Point Entropy  
800 Domain Lena

Table 7.9 shows the total sentinel point information for these two examples.

Domains	Base	Differential	Total
100	361	2641	3002
800	3927	26219	30146

Table 7.9  
Sentinel Point Information for Lena Examples

The sentinel point entropy for both of these examples is below 4 bits/point. For the 100 domain example, quantization can be heavy for all image domains since the piecewise-smooth model of which the sentinel points are a part has significant error before quantization is applied. For the 800 domain example, many domains are quite small and can be quantized heavily without increasing visually perceived distortion. Additionally, the large domains are smoother; the sentinel points of large domains are better predicted by the average predictor. The independent probability estimation contexts for each  $Q_{step}$  take maximum advantage of this size/smoothness disparity.

## 8. Coding Experiments

The preceding chapters have developed machinery for building and representing piecewise smooth image models. Up to this point we have looked at the behavior of each component of this machinery in isolation. We now apply the entire machine to piecewise smooth coding of several Internet images. We give simulated code data for four images: Miss America, Lena, Cameraman, and Baboon that are representative of a wide variety of image types. We then compare the results with the JPEG standard. Finally, we present some sample piecewise-smooth partitions.

### 8.1 Generating Piecewise-Smooth Image Models

Before getting to the data, let's recapitulate the components of the piecewise-smooth image model. An extraction procedure partitions an image into domains, each of which is represented with a polynomial intensity function. The order of the polynomial functions may be zero, one, two, or three, corresponding to one, three, six, or ten coefficients respectively. The desired model order is a parameter of the extraction procedure and is uniform for each image. Of course, domains that contain fewer data points than coefficients have reduced model order. Use of the Cholesky method of Chapter 3 to establish domain order has not been implemented for these simulations. This means that for some smaller domains more sentinel points are transmitted than are needed to recover the domain's intensity function.

The cost function we use for domain extraction is

$$\Delta C = -\Delta MSE + \alpha \cdot \Delta l.$$

The moment methods of Chapter 3 make  $\Delta MSE$  calculations tractable. We use the same cost function for both greedy domain growing and raster-break smoothing. Actually, the cost functions are slightly different since  $\Delta l$  is the change in boundary separators in the cost function for domain growing and is as defined in Table 5.3 for boundary smoothing.



There is as yet no automated mechanism for choosing model extraction and quantization parameters for minimal distortion at a given bit rate. The following parameters are adjusted to give a visually pleasing result at several binary multiples of 100 domains for each test image:

- The global maximum model order.
- The number of domains in the model.
- The amount of noise suppression to apply during extraction.
- The amount of smoothing to apply to the extracted domain boundaries.
- The quantization applied to polynomial coefficients.

Some general proportionality statements can be made about the parameters chosen. As the bit rate decreases:

- Higher model orders are used.
- The extracted model is comprised of fewer domains.
- More noise suppression is used in model extraction.
- More smoothing is applied to domain boundaries.
- Polynomial coefficients are quantized more heavily.

Table 8.1 through Table 8.4 show the parameters used for domain extraction in the coding experiments of Section 8.3. The first column of each table is the overall model order for the image. The second column is the number of domains in each model. The third column is the noise suppression factor used in the greedy domain-growing algorithm. The fourth column is amount of raster-break smoothing applied. In the fifth and sixth columns are the minimum and maximum quantizer step sizes used on the model's sentinel points. The size parameter used to distribute domains into the various quantizers is shown in the last column. The  $|D|_{avg}$  notation in the knee column indicates that the average domain size of the image model was used for  $k$ .

$O_M$	$ P $	$\alpha_n$	$\alpha_s$	$Q_l$	$Q_h$	$k$
3	100	128	1024	16	32	200
3	200	64	768	8	32	100
3	400	8	128	4	32	50
3	800	8	64	2	32	50
2	1600	8	64	4	16	20
1	3200	8	64	4	16	20

Table 8.1  
Domain Extraction Parameters for Lena

$O_M$	$ P $	$\alpha_n$	$\alpha_s$	$Q_l$	$Q_h$	$k$
3	100	64	1024	4	64	$ D _{avg}$
2	200	64	256	4	32	$ D _{avg}$
2	400	64	256	4	32	$ D _{avg}$
2	800	64	256	4	32	$ D _{avg}$
1	1600	64	128	4	32	$ D _{avg}$
0	3200	8	64	2	32	$ D _{avg}$

Table 8.2  
Domain Extraction Parameters for Cameraman

The proportionalities just described are evident in the parameter value data. The exact values of these parameters are not critical and it is seldom necessary to use anything other than integer multiples of base values to achieve visually pleasing results. The default knee parameter used by the sentinel point quantizer is just the model's average domain size. Since domain sizes are not normally uniformly distributed in an image model, a manually tuned (smaller) quantizer knee can often give superior results.

$O_M$	$ P $	$\alpha_n$	$\alpha_s$	$Q_l$	$Q_h$	$k$
3	100	8	128	2	16	200
3	200	8	128	2	16	100
3	400	8	64	2	16	50
3	800	8	64	2	16	50

Table 8.3  
Domain Extraction Parameters for Miss America

Note how the level of quantization decreases as the model fidelity increases. Quantization is heaviest for baboon, where image activity is high and quantization can be coarser before distortion is noticeable. Miss America is quantized lightly at all bit rates shown. This is due to the low-contrast, slowly varying features in the image. Fortunately, since there are few abrupt intensity transitions in this image, prediction works well and low bit rates are attained with relatively little quantization.

$O_M$	$ P $	$\alpha_n$	$\alpha_s$	$Q_l$	$Q_h$	$k$
3	100	1024	1024	16	64	200
3	200	512	512	8	32	100
3	400	256	256	8	32	50
2	800	128	128	8	32	50
2	1600	64	64	4	32	20
1	3200	32	64	2	32	20

Table 8.4  
Domain Extraction Parameters for Baboon

Experience working with piecewise-smooth image models indicates that an image's *average local variance* can be used to automatically establish near optimal values for all parameters except overall model order and domain count. The domain count has a fairly linear relationship to model fidelity. An automatic parameter selection mechanism based upon a user selected overall model order and quality factor (a.k.a. JPEG) certainly seems achievable.

### 8.1.1 Coder Performance

If  $N$  is the number of pixels in the image being coded,  $B$  is the total length of the partition extracted from the image and  $M$  is the number of pixels moved during boundary smoothing, then the complexity bounds for the three main coder components are:

- Greedy domain growing:  $O(N \log^2 N)$
- Raster-break smoother:  $O(B + M)$
- Sentinel point location:  $O(B)$

Since  $B$  and  $M$  are less than  $N$ , the overall complexity is dominated by domain growing. For the 256x256 and 352x288 images in this chapter, domain extraction takes an average of 62 seconds on a 90 MHz Pentium® based personal computer. Smoothing is accomplished in less than 10 seconds. Sentinel points are located in less than 1 second. The extraction times for each image do not differ by more than 20%.

## 8.2 Coding the Image Model

Figure 8.1 is a block diagram of the simulated piecewise-smooth decoder. Domain boundaries of the extracted model are coded using the stroke code of Chapter 6, and polynomial coefficients are coded using the sentinel point code of Chapter 7.

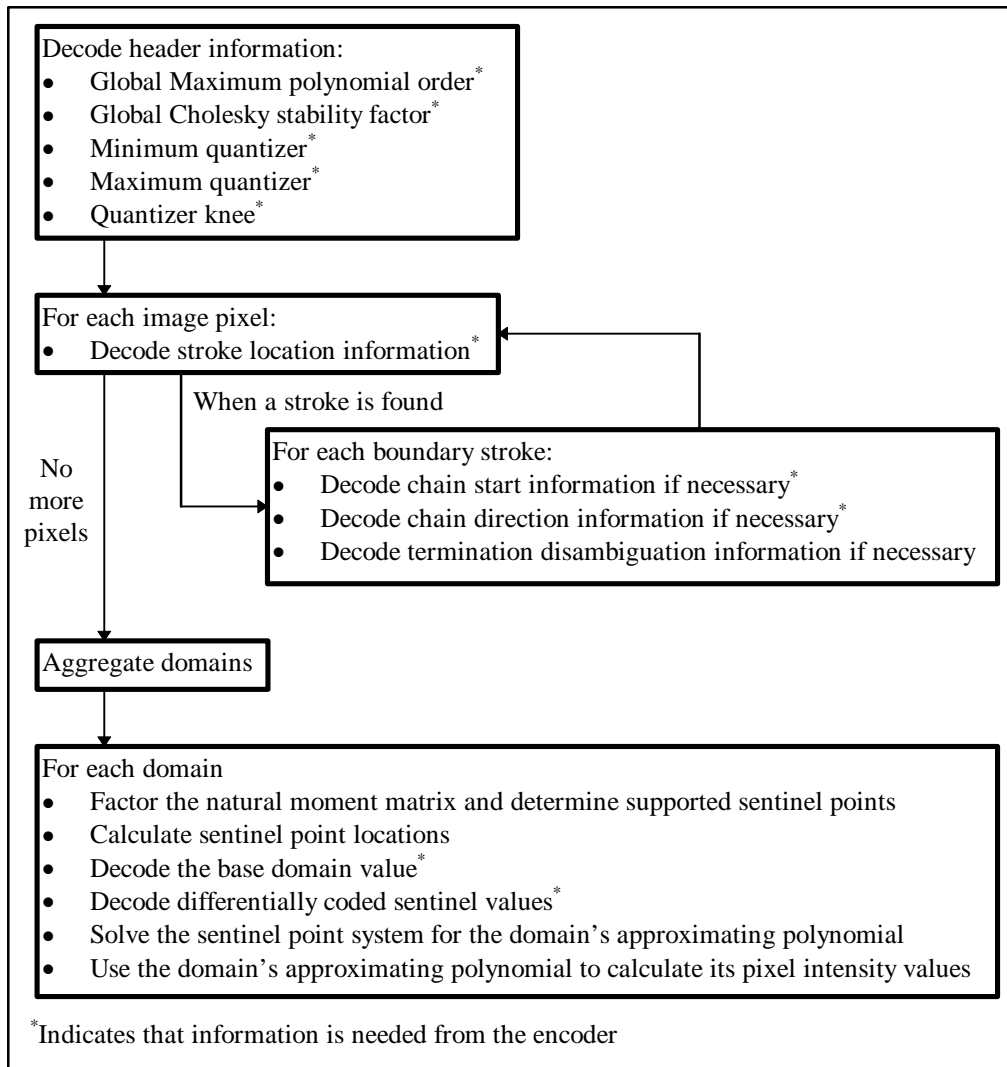


Figure 8.1  
Decoder Functional Diagram

The piecewise-smooth code has very little overhead. Other than stroke and sentinel point information, only five integer values must be transmitted. These global parameters are the maximum polynomial order of the model, the Cholesky stability parameter to use when factoring natural moment matrices, and three variable quantization parameters. The quantization parameters are the minimum and maximum quantizers and the quantizer knee used for decoding sentinel points. After decoding this header information the decoder

commences searching the image in raster order for stroke locations. When one is found it is decoded.

Once all strokes have been decoded, the image pixels are aggregated into domains. The aggregation proceeds by scanning the image in raster order and recursively joining pixels that do not have a boundary separator between them. The order in which domains are encountered in a raster scan determines the order of transmittal of sentinel point values.

Once the domains are established, decoding proceeds one domain at a time. First, a domain's supported sentinel points are found by factoring its natural moment matrix. Second, sentinel point locations are established using the sentinel point location algorithm of Chapter 7. Next, sentinel point values are decoded. After all sentinel points of a domain are decoded, the sentinel system is solved for the domain's polynomial intensity function. Once the polynomial function is available, the domain's pixel intensity values are calculated.

#### *8.2.1 Decoder Performance*

The piecewise-smooth image model is carefully constructed to allow  $O(N)$  decode. The performance bottleneck for the simulated decoder used in these experiments is evaluation of the approximating polynomial for each pixel of the reconstructed image. This operation takes less than 2 seconds on a 90 MHz Pentium® based personal computer for third order models and less than 1 second for simpler models. In an actual decoder, the entropy coder may become the limiting performance factor.

### **8.3 Data**

Table 8.5 is piecewise smooth code data for the Miss America (352x288 pixels) image. Since it has fairly low contrast and few abrupt intensity discontinuities, this image is particularly easy to code for most techniques and the piecewise smooth method does well also. Even at .09 bits/pixel the results are visually pleasing. Note how the boundary information is dominant at the lowest bit rates but becomes less so as the bit rate increases. When the coefficient information becomes dominant, the model order can be reduced to maintain a balance. Model order reduction occurs between the third and fourth lines of the table in this example.

Domains	Order	Boundary Information	Coefficient Information	Total Information	Bit Rate	PSNR
100	3	5317	4037	9354	0.092	37.11
200	3	6710	7693	14402	0.142	38.51
400	3	10397	13964	24361	0.240	39.75
800	2	15077	17029	32106	0.317	40.36
1600	2	23758	29955	53713	0.530	41.40

Table 8.5  
 Piecewise Smooth Code Data  
 Miss America

Code data for a 256x256 pixel Lena is shown in Table 8.6. The 100 domain partition is obviously distinguishable from the original but is visually pleasing nonetheless. The fidelity is unprecedented for a rate of 0.136 bits/pixel on this image. This image contains a large number of moderate contrast features and is quite difficult to code. JPEG at this rate is really nothing but artifacts.

Domains	Order	Boundary Information	Coefficient Information	Total Information	Bit Rate	PSNR
100	3	5911	3001	8912	0.136	26.23
200	3	8165	7158	15323	0.234	28.26
400	3	14903	15438	30340	0.463	31.12
800	3	21181	30145	51326	0.783	33.30
1600	2	29524	35258	64781	0.988	35.22
3200	1	44031	40709	84741	1.293	35.66

Table 8.6  
 Piecewise Smooth Code Data  
 Lena

Table 8.7 contains data for Cameraman. This image has a large amount of high contrast periphery which the piecewise smooth method represents well, but with which JPEG has problems. Again, at 100 domains the rendition is quite pleasing. At 0.293 bits/pixel the only significant distortion is in the grass area, which of course is not efficiently represented by a smooth model. Interestingly, the grass is rendered fairly well starting at 1600 domains.

Domains	Order	Boundary Information	Coefficient Information	Total Information	Bit Rate	PSNR
100	3	6155	2390	8545	0.130	24.50
200	2	9643	3479	13122	0.200	26.39
400	2	12999	6180	19179	0.293	27.28
800	2	16467	12419	28886	0.441	29.03
1600	1	25046	13566	38612	0.589	29.57
3200	0	44022	12554	56576	0.863	32.28

Table 8.7  
 Piecewise Smooth Code Data  
 Cameraman

Table 8.8 is data for Baboon, which is included because it is pathologically non-smooth (it is almost entirely texture). Surprisingly, even Baboon is visually pleasing at 100 domains and at 800 the rendition is becoming quite good. What these results show is that even highly non-smooth features can be rendered if enough domains are used. Since the coefficients of each domain can be quite heavily quantized under these conditions, the representation can be more efficient than may be expected.

Domains	Order	Boundary Information	Coefficient Information	Total Information	Bit Rate	PSNR
100	3	5589	2603	8192	0.125	21.42
200	3	10072	7120	17193	0.262	22.36
400	3	16211	14507	30717	0.469	23.29
800	2	31966	16104	48070	0.733	24.82
1600	2	46183	36033	82216	1.255	26.97
3200	1	61581	38217	99798	1.523	27.69

Table 8.8  
 Piecewise Smooth Code Data  
 Baboon

## 8.4 Sample Partitions

We now present some sample partitions used in our coding experiments. Only the most interesting partitions are included. The 100 domain Lena partition seen previously in Chapters 5, 6, and 7 is not included here. Figure 8.2 shows an interesting partition of Lena's face and is exemplary of the power of the piecewise-smooth extraction method. Figure 8.3 is a 100 domain partition of the cameraman image. The sky in this image is notoriously difficult to



partition. Note the single short boundary segment extending from the head to the top of the image. This strategic behavior is typical.

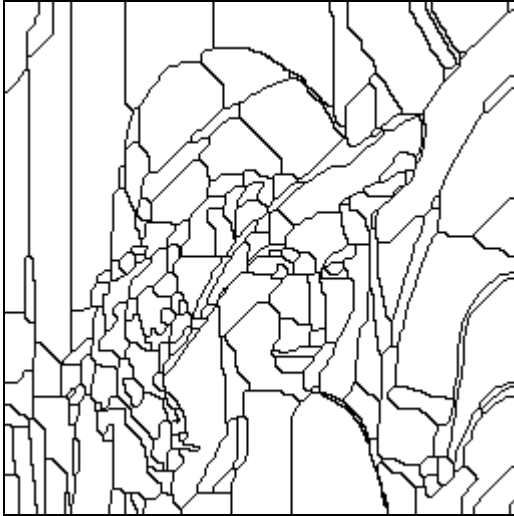


Figure 8.2  
Domain Boundaries  
200 Domain Lena

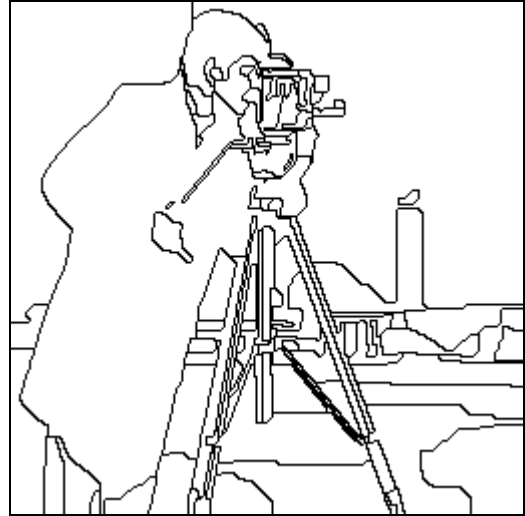


Figure 8.3  
Domain Boundaries  
100 Domain Cameraman

Figure 8.4 is included to show how additional domains fill in the cameraman image. Figure 8.5 shows how domains are allocated to simulate the hair and whiskers of the baboon image



Figure 8.4  
Domain Boundaries  
400 Domain Cameraman

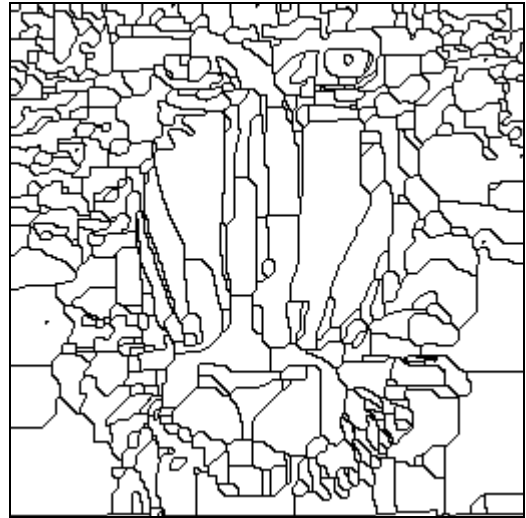


Figure 8.5  
Domain Boundaries  
400 Domain Baboon

Figure 8.6 and Figure 8.7 show the excellent models developed for head and shoulders images. Again note the short boundary segments extending from the head to the top of the image.

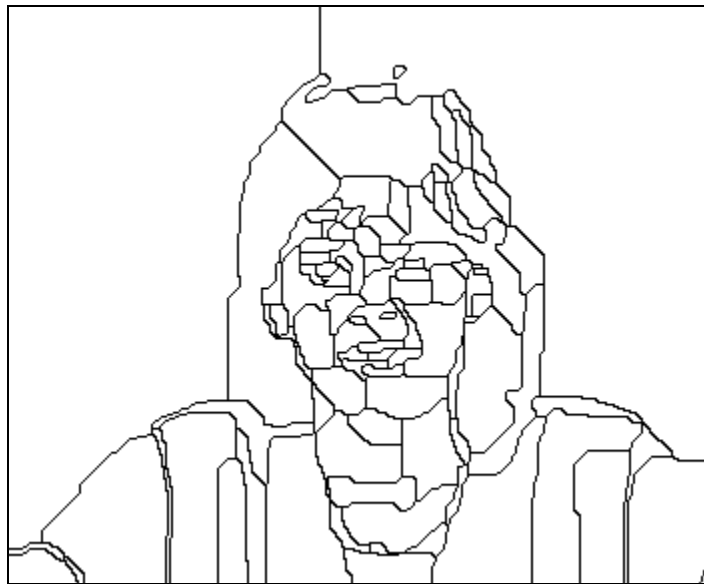


Figure 8.6  
Domain Boundaries  
100 Domain Miss America

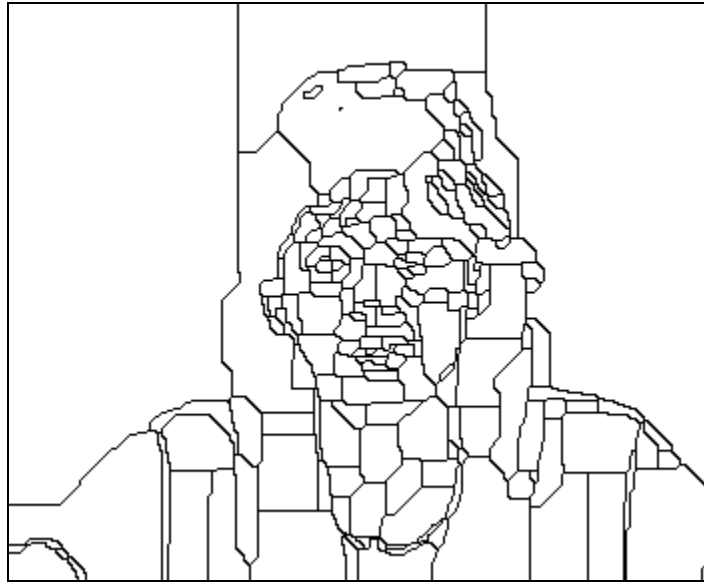


Figure 8.7  
Domain Boundaries  
200 Domain Miss America

### 8.5 JPEG Comparison

Figure 8.8 compares the rate distortion performance of JPEG and the piecewise smooth code on the Miss America image. The piecewise smooth technique is superior up to 0.3 bits/pixel. At this point the only remaining distortion for both techniques is in their rendition of the overlying 2.5 bits of noise. Although the parameter space for the piecewise smooth technique has not been fully searched for higher bit rates on this image, it appears that JPEG may perform better on this noise. Fortunately, noise can easily be replaced by synthetic means at no additional cost.

Figure 8.9 compares both techniques on Lena. Again the piecewise smooth code is superior at low bit rates and remains competitive at higher bit rates. The perceived distortion disparity between the techniques on this image is even greater than the measurements indicate. The piecewise smooth code is doing quite well perceptually at 0.4 bits/pixel whereas noticeable JPEG artifacts do not disappear until near 1 bit/pixel.

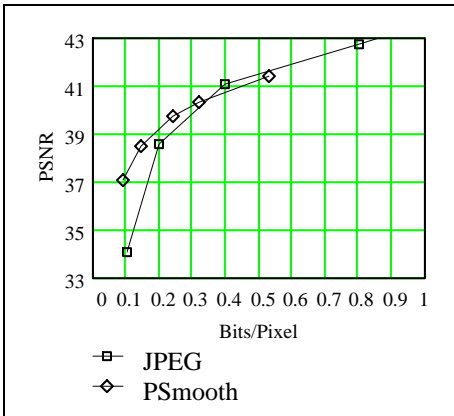


Figure 8.8  
Rate Distortion Comparison  
Miss America

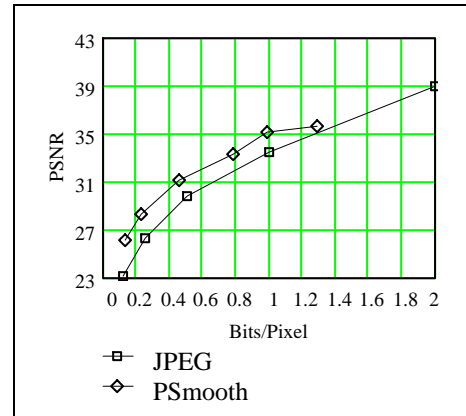


Figure 8.9  
Rate Distortion Comparison  
Lena

Figure 8.10 shows how JPEG never really does get the high contrast edges right. The piecewise smooth Cameraman results are superior to JPEG even at 1 bit/pixel. Figure 8.11 shows the surprising result that the piecewise smooth method is superior to JPEG even on the Baboon image. Again the piecewise smooth technique performs better at low bit rates. The superior performance at higher bit rates is inexplicable.

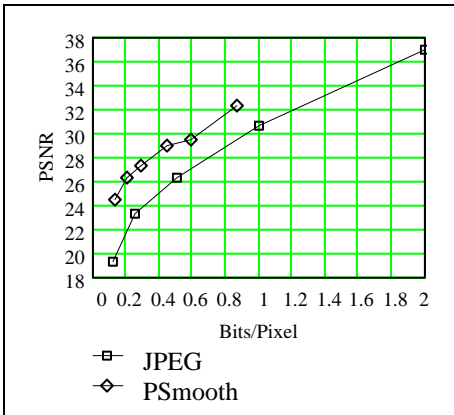


Figure 8.10  
Rate Distortion Comparison  
Cameraman

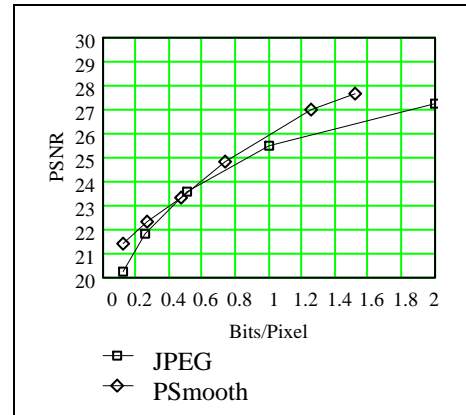


Figure 8.11  
Rate Distortion Comparison  
Baboon

## 8.6 Summary

We applied piecewise-smooth coding to several natural images. We showed the piecewise-smooth model extraction parameters used for each image. We developed a simulated decoder consisting of the stroke code of Chapter 6 and the sentinel point code of Chapter 7 and presented coding results for each image. We compared the piecewise-smooth code to the JPEG lossy image compression standard. Rate/distortion performance of the piecewise-smooth code is superior to JPEG in many instances and is never significantly inferior

The simulated data does not make use of the Cholesky method for determining supported sentinel points. For this reason, some sentinel points are unnecessarily coded. Once this shortcoming is addressed, the bit rate for a given distortion will be reduced. The reduction should be most significant at higher bit rates where image models are comprised of mostly smaller regions.

## 9. Conclusion

### 9.1 Main Contributions

In Chapter 2 we introduce benchmark piecewise-quadratic synthetic images and use them to quantitatively evaluate a piecewise-smooth model extraction algorithm. Both noise-free and noisy benchmarks are presented.

In Chapter 3 we develop moment operators for use in finding least squares piecewise-polynomial approximations of multidimensional data. We introduce terminology for describing the components of multidimensional least squares normal equations. The natural moment matrix is comprised of moments of the independent variables. The forcing moment matrix is comprised of moments of both the independent and dependent variables. We develop complete natural moment matrices, forcing moment matrices and least squares error functions for first, second, and third order two-dimensional polynomials. We extend the Cholesky factorization of symmetric positive definite matrices to symmetric positive semidefinite matrices. We use the modified Cholesky factorization to determine the polynomial coefficients supported by a domain.

In Chapter 4 we develop a general greedy domain growing algorithm for image partitioning that can jointly optimize model complexity and model fidelity. We apply the moment operators and error functions of Chapter 3 to the greedy cost function. If  $N$  is the number of pixels in the partitioned image, the computational complexity of the algorithm is bounded by  $O(N \log^2 N)$ .

In Chapter 5 we develop the raster-break as a formal measure of boundary noise in an image partition and use it to design a state-machine boundary smoothing algorithm. We apply the moment operators and error functions of Chapter 3 to the smoother's cost function. The computational complexity of the smoother is  $O(N)$ .

In Chapter 6 we develop a new chain code for representing the boundaries of a partitioned image. The stroke code is the first code to handle the three direction chain termination

problem on image partitions. We apply the raster drawn boundary criteria of Chapter 5 to develop improved chain direction probability estimation contexts. The stroke code is shown to be superior to raster neighbor codes on sparse image partitions.

In Chapter 7 we develop a method of reconstructing approximating polynomials over arbitrary two-dimensional domains using approximated values at certain geometrically implicit points interior to each domain. We call these points sentinel points. If  $M$  and  $N$  are the maximum extent of a domain in each of its dimensions, we develop  $O(M + N)$  algorithms for finding optimal sentinel points. The number of sentinel points found for a domain corresponds to a global maximum polynomial order and is reduced by the number of unsupported polynomial terms. The Cholesky method of Chapter 3 is the arbiter of a domain's supported polynomial terms. Using the natural moments of the sentinel points and the forcing moments of the original data, we develop an  $O(1)$  algorithm for optimizing the quantized values of the sentinel points. We develop a quality metric to experimentally evaluate the reconstructed polynomial when the sentinel point values are quantized. We develop a method for coding polynomial functions over arbitrary two dimensional domains by applying quantization and differential predictive coding to each domain's sentinel point values. We use a simple global functional dependence to vary the amount of quantization applied to sentinel points based upon a domain's size.

In Chapter 8 we develop a complete code for representing piecewise-smooth image models. The stroke code of Chapter 6 and the sentinel point code of Chapter 7 are the main elements of the code. We evaluate the code on several commonly available images. Rate/distortion performance is superior to JPEG in many instances and is never significantly inferior.

## 9.2 Future Work

The data of Chapter 8 show that the piecewise smooth code is robust across a wide variety of images. At times, it outperforms JPEG by a wide margin and the converse does not appear to be true. What we haven't yet discussed are the weaknesses of the method and possible areas of improvement.

First of all, the piecewise-smooth model extraction parameter space has not been searched systematically. Second, both extraction and coding methods have not been fully analyzed. The biggest remaining hole in the method is the incomplete coding model for sentinel point values.

What's really exciting is that good results have been achieved when there are still many areas of possible improvement. Further, the piecewise-smooth image decomposition is an important new image segmentation algorithm and many related applications can be envisioned. We next discuss just a few of the possible areas for improvement and extension of piecewise-smooth image modeling.

### 9.2.1 Model Extraction

The piecewise-smooth model extraction procedure could be improved in several ways. Using the measures developed in Chapter 4, the noise suppression parameter of the domain extraction algorithm could be made proportional to boundary noise instead of boundary length. Probably the single change that would most improve performance would be to weight model error using *local variance*. The result would be better separation of textured and smooth image areas.

Of larger significance is that the greedy domain growing algorithm is really more appropriate as the middle level of a hierarchical vision system. Local image and domain structure could be used at a lower level to develop a *seed partition* that if used as input to the greedy procedure would lead to a more optimal overall result. Additionally the high memory requirement for complete domain management would be lessened since the number of domains would be reduced prior to invoking the greedy procedure.

### 9.2.2 Model Coding

At the lowest bit rates, boundary information tends to dominate the overall code. An approach that might work to lower boundary information in sparse partitions is *multiscale chain coding*. When coding at very low rates, it is not very clear whether or not just to start from a subsampled image. If a multiscale approach was applied to the boundary code and additional information was needed for an excursion from an even boundary site, a quantitative tradeoff



with subsampling could be made. At higher bit rates, a full investigation needs to be made to see whether additional context information can extend the stroke code to dense partitions or, if not, what other method might be preferred.

The sentinel code simulation does not yet include the capability to systematically exclude coefficients in underdetermined domains that have more pixels than coefficients. Adding this capability in a synergistic way with the extraction procedure would allow for implicit determination of optimal model order for each domain of a partition. Optimal model order could also be made a function of domain size similarly to the size-variable quantization applied to domain sentinel points.

A study of predictors for the sentinel code is needed. Also, the sentinel code does not include prediction information from previously coded adjacent domains. This *inter-domain* information would be most useful for reducing code string length at higher bit rates

### *9.2.3 Extensions*

The most obvious area that needs to be addressed is color, both true color and color maps. Also, optimizations for black/white images could extend the method to more types of images. Texture analysis and synthesis also appear to be directly applicable to the method. Finally, a mechanism for perturbing the model between frames of an image sequence would extend the method to motion estimated video.

## 10. References

---

- <sup>1</sup> David Marr, *Vision*, W. H. Freeman and Company, San Francisco, 1982.
- <sup>2</sup> Robert M. Haralick and Linda G. Shapiro, "SURVEY Image Segmentation Techniques", *Computer Vision, Graphics and Image Processing* 19, 1985, 100-132.
- <sup>3</sup> Y. J. Zhang, "Segmentation Evaluation and Comparison: A study of Various Algorithms", *SPIE Vol. 2094*, 1993, 801-812.
- <sup>4</sup> T. N. Cornsweet, *Visual Perception*, Academic Press, New York, New York, 1970.
- <sup>5</sup> Gilbert Strang, *Linear Algebra and its Applications*, 3<sup>rd</sup> Ed, Harcourt Brace Jovanovich College Publishers, New York, New York, 1988.
- <sup>6</sup> William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, United Kingdom, 1988.
- <sup>7</sup> Stuart Geman and Donald Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, No. 6, November, 1984, pp. 721-741.
- <sup>8</sup> Yvan G. Leclerc, "The Local Structure of Image Intensity Discontinuities", Ph.D. Thesis, McGill University, Montreal, Quebec, Canada, 1989
- <sup>9</sup> Ferran Marques, Antoni Gasull, Todd R. Reed and Murat Kunt, "Coding-Oriented Segmentation Based on Gibbs-Markov Random Fields and Human Visual System Knowledge", *Proc. ICASSP Vol. 4*, Toronto, Canada, May 14, 1991, pp. 2749-2752.
- <sup>10</sup> Jacob Sheinvald, Byron Dom, Wayne Niblack, and David Steele, "Unsupervised Image Segmentation Using the Minimum Description Length Principle", Research Report, IBM Almaden Research Center, 1991.
- <sup>11</sup> Oh-Jin Kwon and Rama Chellappa, "Segmentation-based image compression", *Optical Engineering*, Vol. 32 No. 7, July 1993, 1581-1587.
- <sup>12</sup> P. J. Besl and R. C. Jain, "Segmentation through variable-order surface fitting", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10, 1988, 167-192.
- <sup>13</sup> Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Algorithms*, McGraw-Hill, New York, 1990.
- <sup>14</sup> I Hussain and T. R. Reed, "Compression of still images using segmentation-based approximation", *SPIE Image and Video Compression*, Vol. 2186, 1994, 134-145.
- <sup>15</sup> M. Kass, A. Witkin, and D. Terzopoulos, "SNAKES: Active contour models.", *Int. J. Computer Vision*, Vol. 1, No. 4, 1987, 321-331.
- <sup>16</sup> James D. Foley, Andries van Dam, Steven K. Feiner and John F. Hughes, *Computer Graphics Principles and Practice*, Addison Wesley, Reading, Massachusetts, 1990.
- <sup>17</sup> Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Addison-Wesley, Reading, Massachusetts, 1992.
- <sup>18</sup> Glen G. Langdon, Jr. and Jorma Rissanen, "Compression of Black-White Images with Arithmetic Coding", *IEEE Transactions on Communications*, Vol. COM-29(6), pp. 858-867 (June 1981).
- <sup>19</sup> Robert R. Ester, Jr. and V. Ralph Algazi, "Efficient error free chain coding of binary documents", *Proc. Data Compression Conference*, Snowbird, Utah, March 28, 1995, pp. 122-131.

- 
- <sup>20</sup> Martin J. Turner, "Entropy Reduction via Simplified Image Contourization", NASA Space and Earth Science Data Compression Workshop, Snowbird, Utah, March 27, 1992, pp. 27-42.
- <sup>21</sup> Stephen R. Tate, Lossless Compression of Region Edge Maps, CS-1992-9, Department of Computer Science, Duke University, Durham, NC, 1992.
- <sup>22</sup> C. E. Shannon, "A Mathematical Theory of Communication", The Bell System Technical Journal, Vol. XXVII No. 3, July, 1948, 379-656.
- <sup>23</sup> W. B. Pennebaker, J. L. Mitchell, G. G. Langdon, Jr. and R. B. Arps, "An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder", IBM Journal of Research and Development, Vol. 32(6), pp. 717-726 (November 1988).
- <sup>24</sup> Murat Kunt, Michel Benard and Riccardo Leonardi, "Recent Results in High-Compression Image Coding", IEEE Transactions on Circuits and Systems, Vol. CAS-34, No. 11, November 1987, 1306-1336.