CLASSIFICATION-BASED METHODS IN OPTIMAL IMAGE INTERPOLATION

A Thesis
Submitted to the Faculty

of

Purdue University

by

C. Brian Atkins

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

December 1998

ACKNOWLEDGMENTS

It has been a pleasure to work with my major advisors, Prof. Charles A. Bouman and Prof. Jan P. Allebach, throughout my graduate-level curriculum. I would like to thank them for their high standards; their tireless involvement; their expert guidance and instruction; and their patience. I am also grateful to them for providing the opportunities to work on a variety of interesting projects in the field of digital imaging.

I would also like to convey my gratitude to my family and to Joan for being with me when this task was easy, and when it was difficult. I cannot say how much that has meant to me.

A number of friends in addition to those mentioned above have made this time infinitely more valuable. A complete list of them would have to include Ufuk Agar, Robert delasAlas, Vinay Bhargava, Moses and Linh Chan, Cathy (Cheng-Hua) Chen, Doug and Lisa Gaebler, David and Catherine Lieberman, Steve and Lori Pekarek, Ann Rundell, Srinivas Sista, Lynne Slivovsky, and Chris and Catherine Taylor. I have probably forgotten some others, but I am grateful to God for all of them.

Finally, I would like to thank Prof. Mary E. Bock and Prof. Edward J. Delp for being members of my advisory committee.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

DISCARD THIS PAGE

LIST OF FIGURES

ABSTRACT

Atkins, C. Brian. Ph.D., Purdue University, December 1998. Classification-Based Methods in Optimal Image Interpolation. Major Professors: Charles A. Bouman and Jan P. Allebach.

In this thesis, we introduce two new approaches to optimal image interpolation which are based on the idea that image data falls into different categories or classes, such as edges of different orientation and smoother gradients. Both these methods work by first classifying the image data in a window around the pixel being interpolated, and then using an interpolation filter designed for the selected class. The first method, which we call Resolution Synthesis (RS), performs the classification by computing probabilities of class membership in a Gaussian mixture model. The second method, which we call Tree-based Resolution Synthesis (TRS), uses a regression tree. Both of these methods are based on stochastic models for image data whose parameters must have been estimated beforehand, by training on sample images. We demonstrate that under some assumptions, both of these methods are actually optimal in the sense that they yield minimum mean-squared error (MMSE) estimates of the target-resolution image, given the source image. We also introduce Enhanced Tree-based RS, which consists of TRS interpolation followed by an enhancement stage. During the enhancement stage, we recursively add adjustments to the pixels in the interpolated image. This has the dual effect of reducing interpolation artifacts while imparting additional sharpening. We present results of the above methods for interpolating images which are free of artifacts. In addition, we present results which demonstrate that RS can be trained for high-quality interpolation of images which

exhibit certain characteristic artifacts, such as JPEG images and digital camera images. We also present results of a new interpolative image coding method which uses RS along with the well-known JPEG compression scheme. These results demonstrate that for relatively low bit rates, the RS-based compression scheme can improve upon JPEG compression used alone, in terms of subjective image quality (for an approximately fixed bit-rate), and in terms of better rate-distortion tradeoff.

1. INTRODUCTION

For many applications in image processing, it is necessary to interpolate digital image data in order to render the same image at a higher resolution. An example of such an application is printing, where one has source image data at a fixed resolution, but would like to print it at a higher resolution. Another example is image display, in which one is viewing an image, and would like to zoom in for a better look at a particular detail.

A variety of different approaches to image interpolation have been proposed. Perhaps the most common among them fall into the class of B-spline interpolators [1, 2, 3]. Other methods which are similar include cubic convolution interpolation, which was implemented by Keys [4]. All these methods may be viewed as linear filtering operations.

The main idea behind B-spline interpolation is to specify an interpolating function which approximates the "original" image (which was sampled to obtain the source image data). Since the interpolating function is defined on a continuous-valued parameter, it may be evaluated at arbitrary points. The resulting values comprise the interpolated image. An important requirement is that the interpolating function must pass through the known function values, or source image data. The order in B-spline interpolation refers to the maximum number of continuous derivatives that the interpolating function is required to have. For this reason, higher-ordered B-spline image interpolation results tend to look smoother and more continuous. However, this is at the cost of increased computation.

The simplest of the B-spline interpolators are of the zeroth and first orders. These are known respectively as pixel replication and bilinear interpolation [5]. For these

methods, specification of the interpolating function is trivial. In pixel replication, the interpolating function is piecewise constant. This means that each output pixel simply takes the value of the closest input pixel. In bilinear interpolation, the interpolating function is piecewise linear; it may be obtained by connecting the known data samples with straight lines. This means that each output pixel may be computed as a linear combination of up to four input pixels. Both pixel replication and bilinear interpolation are very common, and they are known to perform satisfactorily for interpolating smooth textures. On the other hand, they offer little in terms of edge and detail rendition.

Third-order, or cubic, B-spline interpolation is next in line with the commonly-used B-spline techniques. In terms of interpolation quality, cubic B-spline interpolation improves upon pixel replication and bilinear interpolation. However, one drawback with cubic B-spline interpolation is that considerable computation is required to specify the interpolating function. For this task, Hou and Andrews [1] have implemented a method which requires the inversion of a large matrix. More recently, Unser, Aldroubi, and Eden [2] have used a more efficient implementation which involves linear filtering. Another drawback of cubic B-spline interpolation is that once the interpolating function has been obtained, still more computation is required to recover the desired output samples.

Each of the algorithms described above amounts to the application of a single linear filter. Since the same filter is applied to obtain each output pixel, these methods tend to average image data across high-contrast boundaries like edges. The result is that interpolations from these methods tend to look blocky or blurry. This is especially true in the cases of pixel replication and bilinear interpolation. In answer to this problem, a wide variety of nonlinear approaches have been proposed.

Edge-directed interpolation algorithms comprise a large subset of these nonlinear approaches. [6, 7, 8, 9, 10]. In edge-directed algorithms, the idea is to find edges in the source data, and to render them continuous and sharp at the target resolution. In local regions where edges are detected, the approach is to find the description of

an edge which passes through the local region. This edge description may be used to interpolate without blurring the values of pixels which lie on different sides of the edge. In regions where edges are not detected, many of the edge-directed algorithms default to bilinear interpolation, since it usually provides very satisfactory results for smooth textures, at minimal computational cost.

One example of such a technique has been implemented by Jensen and Anastassiou [9]. At every pixel where an edge has been detected, they use the pixels in the surrounding 3×3 window to obtain the specification of a continuous-space bilevel edge. This edge specification is used to guide the assignment of pixel values at the target resolution. To specify the edge, it is necessary to determine the orientation and location of the edge (relative to the 3×3 window of pixels), as well as the pixel values for the two levels of the edge.

In contrast to the bilevel model of Jensen and Anastassiou, other models for edge structures have been used. An example is the planar transition model of Algazi, Ford, and Potharlanka [6]. Other approaches to edge-directed interpolation include that of Ayazifar and Lim [8], who have applied their technique to video de-interlacing. More recently, Allebach and Wong [10] have implemented an edge-directed interpolation scheme. In their method, a high-resolution edge map is used to modify bilinear interpolation so that averaging is not performed across boundaries.

Still other nonlinear approaches to image interpolation employ stochastic models for image data. The objective is to obtain a high-resolution rendering which is optimal in some well-defined sense. One such method has been implemented by Schultz and Stevenson [11]. They use a model for image data to define a convex cost functional, which is a function of the high-resolution image. To minimize this cost functional, they employ the method of gradient descent. The result of this minimization is the maximum *a posteriori* (MAP) estimate of the high-resolution image. To define the cost function, they use a model which explicitly represents the acquisition of the source image data. Specifically, they assume that the high-resolution image is the realization of a Huber-Markov random field; and that the low-resolution image was

obtained by block-averaging the high-resolution image, and then adding noise. (They also address the case in which additive noise is not present.) As a starting point for the gradient descent, they use the pixel-replicated interpolation of the low-resolution image.

In this thesis, we introduce two new techniques for optimal image interpolation. Since image data falls into different categories or classes such as edges of different orientation and smooth regions, these methods work by first classifying the image data in a window around the pixel being interpolated, and then using an interpolation filter designed for the selected class. The first method is Resolution Synthesis, which performs classification by computing class membership in a Gaussian mixture model. The second is Tree-based Resolution Synthesis, which uses a binary tree for the classification. We also introduce Enhanced Tree-based RS (ETRS), which consists of TRS interpolation followed by postprocessing to reduce interpolation artifacts while imparting additional sharpening. In addition, we describe a new interpolative image coding method which uses RS along with the familiar JPEG compression scheme. We demonstrate that for relatively low bit rates, the RS-based compression scheme can improve upon JPEG compression used alone, both in terms of subjective image quality (for an approximately fixed bit-rate), and in terms of better rate-distortion tradeoff.

2. RESOLUTION SYNTHESIS

In this chapter, we introduce an interpolation method called Resolution Synthesis (RS). The objective in RS is to compute the minimum mean-squared error estimate of the high-resolution image, given the low-resolution image. For this, we use a stochastic model which explicitly relates vectors containing source image data to corresponding vectors of target-resolution image data. A model which is similar but different from ours has been used by Popat and Picard [12] for image restoration, compression, and generation. Both these models reflect the underlying assumption that image data may be classified into various behaviors or textures. Examples of such behaviors are edges of different orientation, and smooth textures.

The method employed in RS interpolation is to first classify the low-resolution source data; once this has been done, interpolation is performed using a filter which is optimal for the selected class. For example, edges are interpolated using filters which do not smooth out their sharp transitions.

The remainder of this chapter is organized as follows. In Sec. 2.1, we introduce the RS algorithm. To do this, we describe how RS interpolation is carried out, and how the parameters for the RS model are obtained. In Sec. 2.2, we describe an efficient implementation of the RS algorithm. In Sec. 2.3 we describe our extension of the RS algorithm for interpolation of color images, and we also describe how we can build sharpening in to the RS interpolation procedure. In Sec. 2.4, we present results of RS interpolation. In Sec. 2.5, we demonstrate that the RS interpolator parameters can be trained for high-quality interpolation of images such as JPEG images and digital camera images, which exhibit certain characteristic artifacts. Finally, we present conclusions in Sec. 2.6.

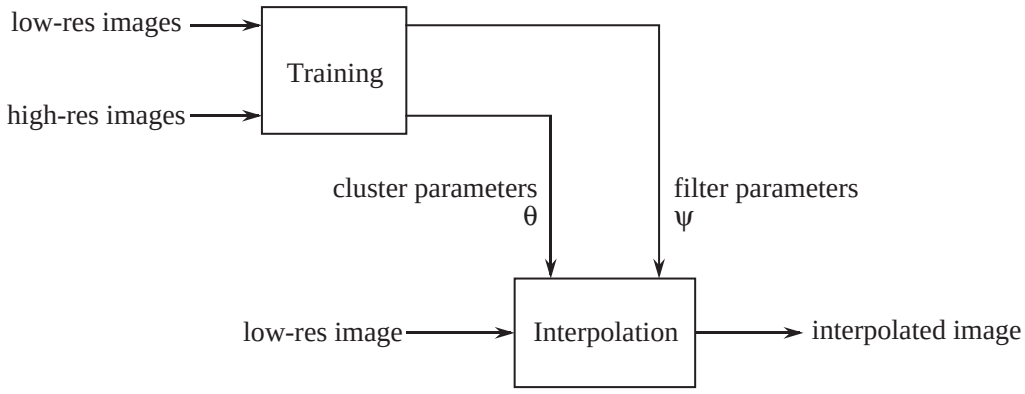


Fig. 2.1. Overview of Resolution Synthesis.

2.1 Derivation of the RS Algorithm

The following subsections present an analytical derivation of the RS model. We will show that, under the proper assumptions, the RS algorithm is the minimum mean squared error estimator of the desired interpolated image given the low-resolution image. However, the specific form of the interpolation will depend on the model parameters, which must be precomputed in a separate training process. Fig. 2.1 illustrates how the training and interpolation algorithms are used. The training process is first run off-line on a large database of representative low- and high- resolution images. We will see that this training process is implemented using an analytical technique known as the expectation-maximization (EM) algorithm.

The training process has two outputs: the interpolation filters, which we denote as ψ, and the cluster parameters, which we denote as θ. We will see that ψ represents a set of interpolating filters, with one filter for each class; while θ defines the characteristics of each class, and how to separate them.

2.1.1 Interpolator structure

Fig. 2.2 gives a more detailed view of the structure of the interpolator. The function of the interpolator in this example is to replace a single low-resolution pixel with 4 high-resolution pixels, which we will denote by the vector x. More generally,

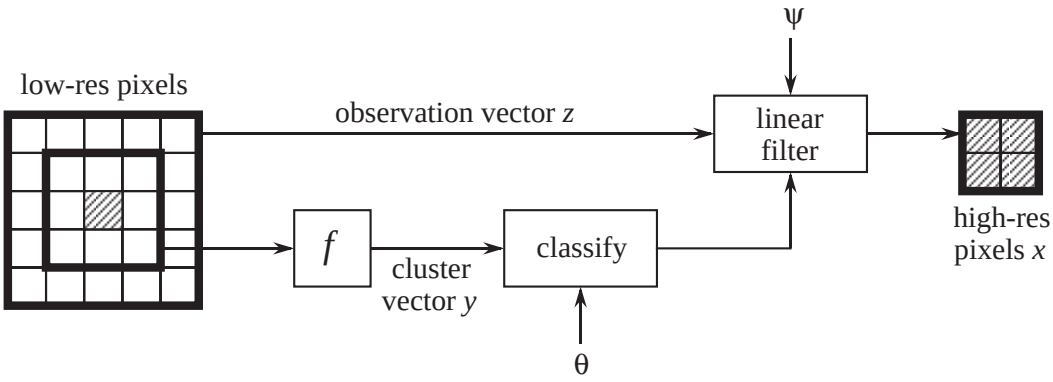


Fig. 2.2. Structure of the interpolator.

for interpolation by a factor L, the low-resolution pixel would be replaced with L^2 pixels.

The interpolator works by using the low-resolution pixels in a 5×5 window centered about the pixel to be replaced. We will refer to this window of pixels as the *observation vector, z*. The interpolation is performed by first classifying the windowed region, and then interpolating using the selected filter. Later we will see that this classification is actually soft, and in general, more than one filter may be applied. In this case, the filter outputs are combined in proportions which reflect the probabilities that the region is from the selected classes.

An important fact is that the region used for classification is smaller than that used for filtering. We will refer to the data extracted in this smaller 3×3 window as the *cluster vector, y*. In general, y and z will be related by a nonlinear function $f(\cdot)$:

$$y \;=\; f(z)\;. \tag{2.1}$$

In this report, $f(\cdot)$ depends only on the 3×3 center region of the window; however, the analytical results we will present do not depend on this assumption.

We will see that the specific choice of the function $f(\cdot)$ can be very important in determining the quality of the interpolation. This is because $f(\cdot)$ may be chosen to nonlinearly distort the space of the cluster vectors, and thereby accentuate the features of greatest importance. For example, rendition of edges is of great importance

in image interpolation. By careful selection of the function $f(\cdot)$, we can accentuate edges so that they are better separated from other image features.

The analysis will require some additional definitions. Let u be the part of the observation vector, z, that is not represented in the cluster vector, y. Formally, we say that there exists a one-to-one vector-valued function g, such that $(u, y) = g(z)$. Since we will be modeling both the low- and high- resolution images as random quantities, we will follow the convention of using upper-case letters for random variables and lower-case letters for their realizations.

2.1.2 Optimal interpolation

In this section, we will derive a minimum mean-squared error interpolator, or equivalently, a minimum mean-squared error predictor for the high-resolution image given the low-resolution image. First, we will assume that we know the distribution of the images, but in later sections we will derive the formulas for the training algorithm by which we estimate the parameters of these distributions.

All of our analysis will be based on three assumptions about the image data.

Assumption 1: The cluster vector Y is distributed as a multivariate Gaussian mixture. That is, the probability density for $p_y(y)$ has the form

$$p_y(y) \;=\; \sum_{j=1}^{M} p_{y|j}(y \mid j)\pi_j \;, \tag{2.2}$$

where $p_{y|j}(y \mid j)$ is a multivariate Gaussian density for each j, and π_j is the probability of class j.

This assumption is not very strong, since we may closely approximate the true distribution by using a large number M of Gaussian components. We will denote the random variable corresponding to the specific component of the mixture as J, where J takes discrete values between 1 and M, with probability π_j. Intuitively, J represents the class at each pixel. Since the Gaussian components will generally overlap, it is impossible to exactly determine the class to which a pixel belongs. However, we may still think of J as being an unobserved random quantity.

Assumption 2: The conditional distribution of X given Z and J is multivariate Gaussian, with mean $A_J Z + \beta_J$, and covariance Λ_J. We write this assumption as

$$X \mid Z, J \;\sim\; N(A_J Z + \beta_J, \Lambda_J) \; . \tag{2.3}$$

We will see that the filter parameters specified by ψ are obtained directly from the parameters for these distributions, so we write

$$\psi \;=\; (\{A_j\}_{j=1}^M, \{\beta_j\}_{j=1}^M, \{\Lambda_j\}_{j=1}^M) \; . \tag{2.4}$$

This assumption is similar to the previous one since it requires that a sufficiently large number of classes be chosen so that for each class, J, the high-resolution pixels are distributed as a linear transformation of the low-resolution pixels plus multivariate Gaussian noise.

Assumption 3: The class J is conditionally independent of the high- and low-resolution vectors X and Z, given the cluster vector Y. Formally, this means that

$$p_{j|x,z}(j \mid x, z) \;=\; p_{j|y}(j \mid y) \; . \tag{2.5}$$

Intuitively, this assumption means that the cluster vector Y gives us all possible information required to determine the class J. This is a strong assumption since Z contains additional information which might be useful in determining the class. In addition, the high resolution pixels could be valuable themselves in identifying the true class. However, we make this assumption since it will lead to important algorithmic simplifications that we have found improve interpolation quality.

Given these assumptions, we may compute the minimum mean squared error [13] interpolator as

$$\hat{X} \;=\; E[X \mid Z] \tag{2.6}$$

$$\;=\; \sum_{j=1}^{M} E[X \mid Z, J = j] p_{j|z}(j \mid Z) \tag{2.7}$$

$$\;=\; \sum_{j=1}^{M} (A_j Z + \beta_j) p_{j|y}(j \mid Y) \; . \tag{2.8}$$

To obtain the first and second terms in the argument of the last summation, we have invoked assumptions 2 and 3, respectively.

In order to compute $\hat{X}$, we must be able to compute $p_{j|y}(j \mid y)$, the probability that the cluster vector y belongs to class j. By assumption 1, the conditional probability of Y given J is a multivariate Gaussian. We will denote the mean of each Gaussian distribution to be μ_j. In general, the covariance could be chosen to vary with each class j. However, we have found that choosing the covariance for all classes to be $\sigma^2 I$, where I is the identity matrix, has many advantages. First, we have found that this restriction actually improves the quality of the interpolated image. While this may seem counter-intuitive, it is quite reasonable, since the reduced number of parameters makes training more effective and accurate. Second, we will see that this diagonal covariance leads to much simpler computations by eliminating off-diagonal terms.

We will denote the parameters of $p_y(y)$ as θ. Since the distribution for Y is assumed to be a Gaussian mixture model,

$$\theta = \left\{ \{\mu_j, \pi_j\}_{j=1}^{M}, \sigma \right\} , \tag{2.9}$$

where μ_j is the mean vector for class j, π_j is the prior probability for class j, and σ is the variance for all classes. This leads to the density function

$$p_y(y|\theta) = \sum_{j=1}^{M} \frac{\pi_j}{(2\pi\sigma^2)^{d/2}} \exp\left(\frac{-1}{2\sigma^2} \|y - \mu_j\|^2 \right) , \tag{2.10}$$

where d is the dimension of the vector y. Using this result and Bayes' rule, we may compute the probability $p_{j|y}(j \mid y, \theta)$

$$p_{j|y}(j \mid y, \theta) = \frac{p_{y|j}(y \mid j, \mu_j, \sigma)\pi_j}{p_y(y \mid \theta)} \tag{2.11}$$

$$= \frac{\exp\left(\frac{-1}{2\sigma^2} \|y - \mu_j\|^2 \right) \pi_j}{\sum_{l=1}^{M} \exp\left(\frac{-1}{2\sigma^2} \|y - \mu_l\|^2 \right) \pi_l} . \tag{2.12}$$

Inserting (2.12) into (2.8), we obtain the equation for optimal interpolation, in terms of the unknown parameters:

$$\hat{X} = \sum_{j=1}^{M} (A_j Z + \beta_j) \frac{\exp\left(\frac{-1}{2\sigma^2} \|y - \mu_j\|^2 \right) \pi_j}{\sum_{l=1}^{M} \exp\left(\frac{-1}{2\sigma^2} \|y - \mu_l\|^2 \right) \pi_l} . \tag{2.13}$$

The optimal interpolation is computed as a combination of the outputs of all M of the interpolating filters. This is illustrated in Fig. 2.3. Since the model must have enough classes to represent a wide variety of textures, this may require excessive computation. However, we will see that in terms of interpolation quality, much of this computation is not necessary. We will show that with a simple and efficient method for filter selection, we can achieve virtually equivalent interpolation quality, by using an average of only 1 or 2 interpolating filters for each pixel.

2.1.3 Parameter estimation for the interpolator

In the preceding section, we derived the equation for optimal interpolation assuming that we knew the model parameters θ and ψ. However, it is unrealistic to assume that we will know these parameters *a priori*, so we must estimate them from example high and low resolution images. This training step is computationally intensive, but since it can be performed off-line, it does not impact the speed of the RS interpolation.

In this section, we give a detailed description of the training procedure required to estimate the model parameters. The parameters are computed from a training database which consists of pairs of low- and high-resolution images. Of course, our choice of the training set will affect the appearance of the interpolated images. While this may seem like a limitation of RS for use in a general context, we will see that this may be turned into an advantage, since the training set can be designed for interpolation of specific types of image data.

Our approach will be to compute maximum likelihood (ML) estimates [14, 13] of the required parameters ψ and θ given the set of high and low resolution data. Specifically, our observations will be realizations of X_s, U_s, and Y_s, where s denotes a pixel in a large set S of pixels at which we obtain training vectors. Notice that we use U_s and Y_s in place of Z_s since (U_s, Y_s) contains the same information as Z_s but results in a simpler analytical form for the result. The ML estimates for ψ and θ are

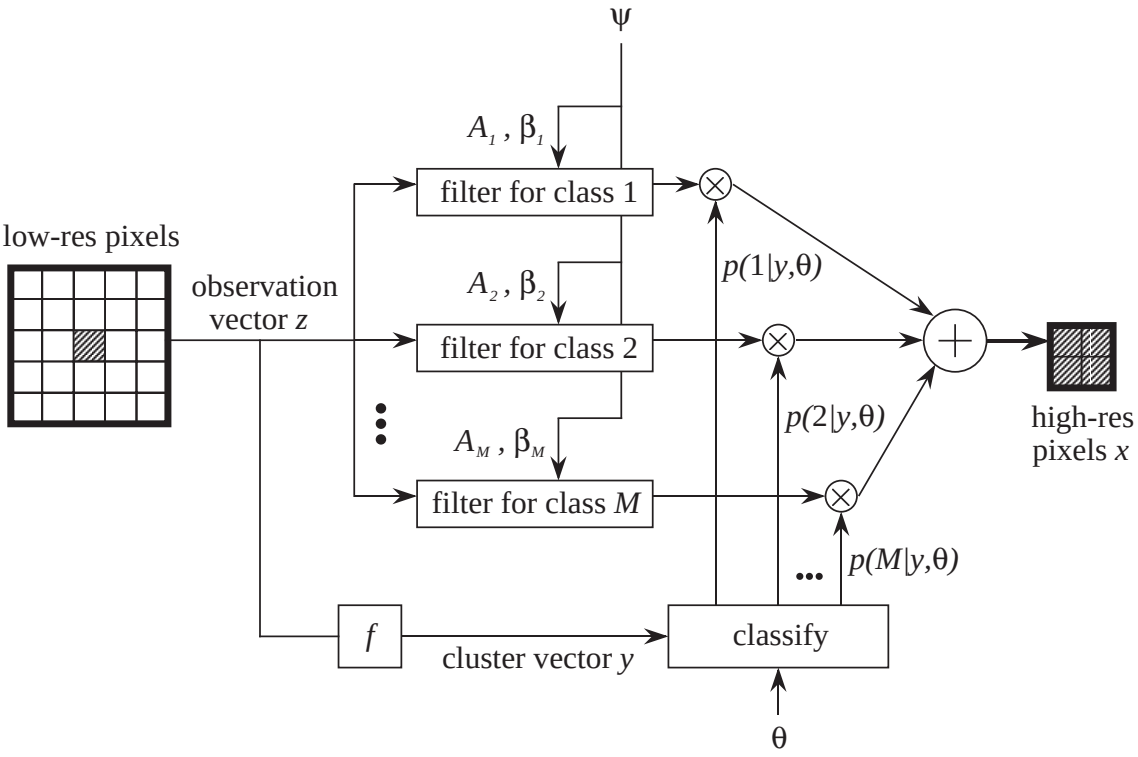


Fig. 2.3. Detailed illustration of the RS interpolation process.

then given by

$$\psi^*, \theta^* \quad = \quad \arg\max_{(\psi,\theta)} \prod_{s \in S} p_{x,u,y}(x_s, u_s, y_s \mid \psi, \theta) \tag{2.14}$$

where we assume that the observations (X_s, U_s, Y_s) are independent for each pixel $s \in S$.

Computation of ψ^*, θ^* is generally difficult because we do not have the class, J, of each pixel available. Instead, we must estimate the parameters using only X, U, and Y. This is known as the incomplete data problem where the set X, U, Y, J are known as the complete data.

There are a variety of methods for addressing this problem, but the most widely used is the expectation-maximization (EM) algorithm [15, 16, 17]. The EM algorithm works by using the log likelihood expression of the complete data, but taking the expectation over the unknown value J. The maximization is then repeatedly performed over this modified likelihood. More specifically, the EM algorithm is generally given by the following iterations in k

$$\psi^{(k+1)}, \theta^{(k+1)} \quad = \quad \arg\max_{(\psi,\theta)} Q(\psi, \theta \mid \psi^{(k)}, \theta^{(k)}) \tag{2.15}$$

where the Q function is defined by the expression

$$Q(\psi, \theta \mid \psi^{(k)}, \theta^{(k)}) = \tag{2.16}$$
$$E\left[\log \prod_{s \in S} p_{x,u,y,j}(x_s, u_s, y_s, J_s \mid \psi, \theta) \mid X_s = x_s, U_s = u_s, Y_s = y_s, \psi^{(k)}, \theta^{(k)}\right] .$$

An important property of the EM algorithm is that with every iteration, the likelihood is increased. So for each k, we have

$$\prod_{s \in S} p_{x,u,y}(x_s, u_s, y_s \mid \psi^{(k+1)}, \theta^{(k+1)}) \geq \prod_{s \in S} p_{x,u,y}(x_s, u_s, y_s \mid \psi^{(k)}, \theta^{(k)}) . \tag{2.17}$$

This implies that if the log likelihood sequence has an upper bound, then the sequence will converge to at least a local maximum of the likelihood function [16, 17].

By assumption 3, we know that $p_{j|x,z}(j \mid x, z) = p_{j|y}(j \mid y)$. Since $p_{j|y}(j \mid y)$ only depends on θ, this leads to the following simplification.

$$Q(\psi, \theta \mid \psi^{(k)}, \theta^{(k)}) = \tag{2.18}$$

$$
\begin{aligned}
& E\left[\sum_{s\in S}\log p_{x,u,y,j}(x_s,u_s,y_s,J_s \mid \psi,\theta) \mid X_s=x_s, U_s=u_s, Y_s=y_s, \psi^{(k)}, \theta^{(k)}\right] \\
= \;& E\left[\sum_{s\in S}\log p_{x,u,y,j}(x_s,u_s,y_s,J_s \mid \psi,\theta) \mid Y_s=y_s, \theta^{(k)}\right] && (2.19)\\
= \;& Q(\psi,\theta \mid \theta^{(k)}) \; . && (2.20)
\end{aligned}
$$

Next observe that

$$
\begin{aligned}
& \log p_{x,u,y,j}(x,u,y,J \mid \psi,\theta) = && (2.21)\\
& \qquad \log p_{x|z,j}(x \mid z,J,\psi) + \log p_{u|y,j}(u \mid y,J) + \log p_{y,j}(y,J \mid \theta) \; ,
\end{aligned}
$$

where we note that conditioning on (U,Y) is equivalent to conditioning on Z. Using this relationship, we may break the expression for Q into three parts

$$
Q(\psi,\theta \mid \theta^{(k)}) \;=\; Q_1(\psi \mid \theta^{(k)}) + Q_2(\theta \mid \theta^{(k)}) + \mathrm{C}(\theta^{(k)}) \; , \qquad (2.22)
$$

where

$$
\begin{aligned}
Q_1(\psi \mid \theta^{(k)}) &= \sum_{s\in S} E\left[\log p_{x|z,j}(x_s \mid z_s, J_s, \psi) \mid Y_s=y_s, \theta^{(k)}\right] \; , && (2.23)\\
Q_2(\theta \mid \theta^{(k)}) &= \sum_{s\in S} E\left[\log p_{y,j}(y_s, J_s \mid \theta) \mid Y_s=y_s, \theta^{(k)}\right] \; , && (2.24)
\end{aligned}
$$

and

$$
\mathrm{C}(\theta^{(k)}) = \sum_{s\in S} E\left[\log p_{u|y,j}(u_s \mid y_s, J_s) \mid Y_s=y_s, \theta^{(k)}\right] \; . \qquad (2.25)
$$

This simplified form for Q allows us to simplify the EM updates. In particular, notice that while Q_1, Q_2, and C all depend upon $\theta^{(k)}$, only Q_2 depends upon θ. This means that the values of $\theta^{(k)}$ may be computed through iterative maximization of Q_2 only. Once the iterations converge to a final estimate $\hat{\theta}$, the final value of ψ may be computed by maximizing Q_1 with respect to ψ. This results in the simplified EM update equations

$$
\begin{aligned}
\theta^{(k+1)} &= \arg\max_{\theta} Q_2(\theta \mid \theta^{(k)}) && (2.26)\\
\hat{\theta} &= \lim_{k\to\infty} \theta^{(k)} && (2.27)\\
\hat{\psi} &= \arg\max_{\psi} Q_1(\psi \mid \hat{\theta}) \; . && (2.28)
\end{aligned}
$$

Because the maximization for θ amounts to determination of the parameters for the distribution of cluster vectors, we refer to the iteration of (2.26) as *clustering*. The maximization for ψ is the computation of the parameters for the interpolating filters, so we refer to that procedure as *filter design*.

To determine the update equation of (2.24), we may write Q_2 explicitly as the following sum:

$$\begin{aligned} Q_2(\theta \mid \theta^{(k)}) &= \sum_{s\in S}\sum_{j=1}^{M} (\log p_{y,j}(y_s, j \mid \theta))\, p_{j|y}(j \mid y_s, \theta^{(k)}) && (2.29)\\ &= \sum_{s\in S}\sum_{j=1}^{M} p_{j|y}(j \mid y_s, \theta^{(k)}) \log p_{y|j}(y_s \mid j, \mu_j, \sigma) && (2.30)\\ &\quad + \sum_{s\in S}\sum_{j=1}^{M} p_{j|y}(j \mid y_s, \theta^{(k)}) \log \pi_j \ . \end{aligned}$$

We note that Q_2 separates into two terms. The first term only depends on $\{\mu_j\}_{j=1}^{M}$ and σ^2; and the second term only depends on $\{\pi_j\}_{j=1}^{M}$. These terms may be maximized for their respective parameters separately. The required maximizations are easy to derive [18], and they result in the update equations for clustering:

$$\begin{aligned} N_j^{(k+1)} &= \sum_{s\in S} p_{j|y}(j \mid y_s, \theta^{(k)}) \ ; && (2.31)\\ \pi_j^{(k+1)} &= \frac{N_j^{(k+1)}}{N} \ ; && (2.32)\\ \mu_j^{(k+1)} &= \frac{1}{N_j^{(k+1)}} \sum_{s\in S} y_s p_{j|y}(j \mid y_s, \theta^{(k)}) \ ; && (2.33) \end{aligned}$$

and

$$\sigma^{2^{(k+1)}} = \frac{1}{d}\sum_{j=1}^{M}\left[\pi_j^{(k+1)} \frac{1}{N_j^{(k+1)}} \sum_{s\in S} \|y_s - \mu_j^{(k+1)}\|^2 p_{j|y}(j \mid y_s, \theta^{(k)})\right] \ , \quad (2.34)$$

where d is the dimension of the cluster vector.

To begin clustering, we must specify an initial value for θ. The initialization for the mean vectors $\{\mu_j\}_{j=1}^{M}$ of the clusters is the most important aspect of this initialization, since they define the initial classes which are "starting points" for clustering. We initialize the means with cluster vectors from the training set. To encourage the

formation of distinct classes, we use cluster vectors from different images (if there are multiple images in the training set); and if multiple cluster vectors are used from a single image, they are chosen from spatially separate regions of that image. For the probabilities $\{\pi_j\}_{j=1}^{M}$ of the classes, we assign equal probabilities of $\frac{1}{M}$. The variance σ^2 is initialized as

$$\sigma^{2^{(0)}} \;=\; \frac{1}{d}\sum_{i=1}^{d} r_i \;, \tag{2.35}$$

where r_i is the sample variance of the i–th element of the cluster vectors defined by

$$r_i \;=\; \frac{1}{N}\sum_{s\in S}(y_{s\,i} - \overline{y_{s\,i}})^2 \;. \tag{2.36}$$

Here we use the notation $y_{s\,i}$ to denote the i^{th} component of the vector y_s. With an initial estimate specified, clustering proceeds through iterative evaluation of the update equations (2.31)–(2.34). We stop iterating through the update equations when the log likelihood ceases to increase appreciably from one iteration to the next. At this point, we regard the resulting estimate $\hat{\theta}$ as the final estimate of θ.

Once clustering is complete, we design the interpolating filters, which amounts to estimation of ψ through (2.28). A derivation of this maximization is included in the appendix. Essentially, we obtain the solution by finding the roots of the derivatives of Q_1, with respect to its parameters. For this, we define the following weighted sample statistics:

$$N_j \;\stackrel{\text{def}}{=}\; \sum_{s\in S} p_{j|y}(j \mid y_s, \hat{\theta}) \;; \tag{2.37}$$

$$\nu_j \;=\; \begin{pmatrix} \nu_{x|j} \\ \nu_{z|j} \end{pmatrix} \tag{2.38}$$

$$\stackrel{\text{def}}{=}\; \frac{1}{N_j}\sum_{s\in S}\begin{pmatrix} x_s \\ z_s \end{pmatrix} p_{j|y}(j \mid y_s, \hat{\theta}) \;; \tag{2.39}$$

and

$$\Gamma_j \;=\; \begin{pmatrix} \Gamma_{xx|j} & \Gamma_{xz|j} \\ \Gamma_{xz|j}^t & \Gamma_{zz|j} \end{pmatrix} \tag{2.40}$$

$$\stackrel{\text{def}}{=} \quad \frac{1}{N_j} \sum_{s \in S} \begin{pmatrix} x_s - \nu_{x|j} \\ z_s - \nu_{z|j} \end{pmatrix} \left((x_s - \nu_{x|j})^t \, (z_s - \nu_{z|j})^t \right) p_{j|y}(j \mid y_s, \hat{\theta}) \; ; \quad (2.41)$$

for $1 \leq j \leq M$. The maximization required by (2.28) is obtained as

$$\hat{A}_j \;\; = \;\; \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \; ; \quad (2.42)$$

$$\hat{\beta}_j \;\; = \;\; \nu_{x|j} - \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \nu_{z|j} \; ; \quad (2.43)$$

and

$$\hat{\Lambda}_j \;\; = \;\; \Gamma_{xx|j} - \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \Gamma_{xz|j}^t \; ; \quad (2.44)$$

for $1 \leq j \leq M$. We use (2.42) and (2.43) to compute the filter parameters, and (2.44) provides estimates of the error variance for interpolation, within the classes.

We remark that if the estimate $\hat{\theta}$ from clustering equals its true ML value of θ^*, then the filter parameters obtained through (2.42), (2.43) and (2.44) also equal their true ML value of ψ^*. But we can not guarantee that $\hat{\theta}$ is a true ML estimate, since this generally depends upon the initial conditions for clustering. However, we will see that by following the procedure described above, we have obtained very satisfactory results.

One aspect of the algorithm which we should discuss is our choice of the number M of classes of image data. More experimentation is needed before we can have a full understanding of this matter; but for various cases, we have achieved high-quality interpolation with model orders of around $M = 100$. Strictly speaking, the order is a parameter which should be identified along with the other parameters of the model, under the same criterion of optimality. A model with too few classes would effectively under-represent edge classes, which would yield excessively smooth interpolation. A model with too many classes would "over-fit" the model to our training data, which would degrade interpolation in a general context.

Theoretical approaches to order identification include the so-called "penalized" likelihood criteria of Akaike and Rissanen [19, 20]. Each of these criteria includes a term for the likelihood of the model, which is weighed against a term for the order

of the model, so that models with excessively low or high orders are discouraged. (In cases where the model order is fixed, both of these criteria reduce to the familiar ML criterion.) The criterion of Rissanen, which is known as the *description length*, proposes that the best model is the one with which we would require a minimum of bits to rewrite the observed data. The description length has the advantage that it has been shown to yield a consistent estimator in contexts where that of Akaike does not [21]. To make use of the criterion of Rissanen, we have employed an agglomerative clustering strategy, which is described more thoroughly in [22]. Here, we estimate several models of different orders, and select for our model that which was best under the description length criterion. This strategy is agglomerative, in the sense that we estimate models for a decreasing sequence of orders. We begin by estimating a model with a maximum allowable order. Next, we use the current model to create a new model, by combining two of the classes in the model; and we use that model as an initial condition for clustering, which yields a new model estimate (for the new lower order). We continue in this fashion until models for all orders less than the maximum have been estimated. When this procedure is complete, we select that model which has the lowest description length.

We have tried this with a variety of types of data. However, we have found that in almost all cases, the model with the maximum allowable number of classes (or a number close to it) has the lowest description length. We have found that models with orders of around $M = 100$ have enough classes to efficiently represent a wide variety of textures.

2.1.4 Projection operator $f(\cdot)$ for cluster vectors

In this subsection, we describe the projection operator $f(\cdot)$ from (2.1). This function is used to obtain the cluster vector y from the low-resolution observation vector z. Selection of this function is of critical importance, since it ultimately affects how well RS renders edges and detail at the target resolution.

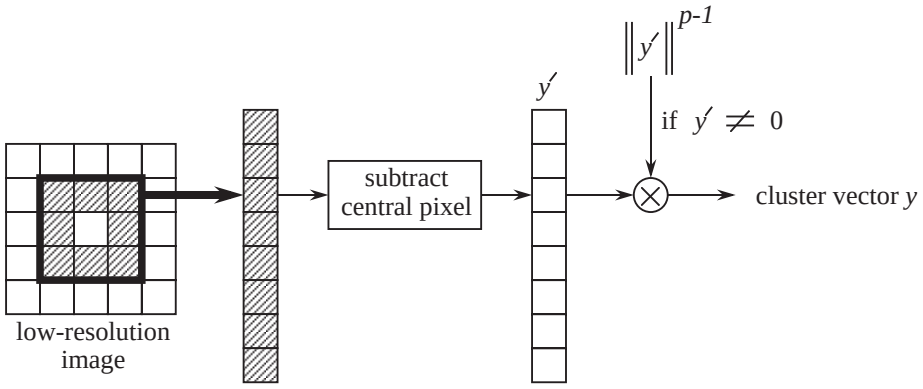


Fig. 2.4. Construction of a cluster vector.

To define the function $f(\cdot)$, it will be convenient to introduce the vector y', which is a lower-dimensional projection of z. To construct y' we stack the 8 pixels adjacent to the central pixel into a vector, and then we subtract the value of the central pixel from each of these elements. Then $f(z)$ is evaluated by scaling y':

$$f(z) \;=\; \begin{cases} y' \, \|y'\|^{p-1} & \text{if } y' \neq 0 \\ 0 & \text{else} \end{cases} \quad . \tag{2.45}$$

Here, p is a scalar which may take any value between 0 and 1; we have obtained our best results by using a value of $p = \frac{1}{4}$. The whole process of evaluating the function $f(\cdot)$ from the observation vector z is illustrated in Fig. 2.4.

To see how the projection operator affects the performance of RS, it is important to remember two things. First, the vector y' actually contains the differences between the central pixel and its 8 nearest neighbors. This means that in terms of Euclidean distance, cluster vectors taken from smooth textures will be concentrated near the origin, while cluster vectors taken from edges will be farther away from the origin. Second, the scaling of (2.45) modifies the length of y' while preserving its direction. Geometrically the effect of this scaling is to warp the space of cluster vectors so that edges of different magnitude but similar orientation or shape are grouped together. The result is that edge and detail classes can be accentuated during clustering, which allows for better discernment of such behaviors.

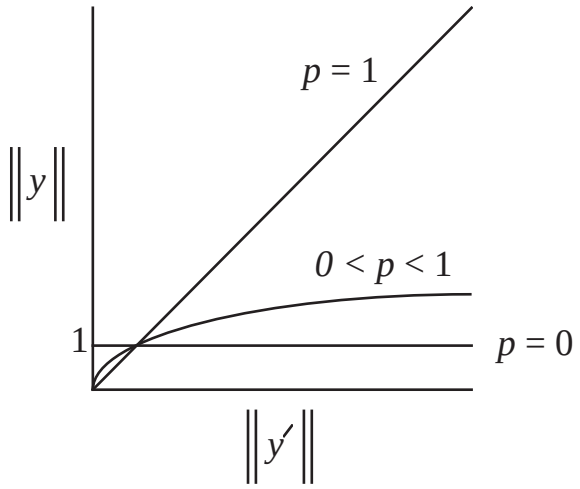


Fig. 2.5. Scaling transformation for the projection operator $f(\cdot)$, for various values of p.

We have found that the value of the scalar p, which specifies the scaling transformation of (2.45), exerts considerable influence over how classes are defined during clustering. To see why this is, compare Fig. 2.5 with the scaling transformation of (2.45). By using $p = 1$ in the scaling transformation, we are effectively leaving the vector y' unchanged. In this case, the histogram of cluster vectors is concentrated near the origin. The result is that during clustering, relatively few classes are established for edges and detail, so that these behaviors can not be rendered accurately. On the other hand, by using a value of $p = 0$, we project every nonzero cluster vector onto the sphere of unit radius. During clustering, nearly every class that is established represents some type of edge or detail. While this improves performance with edges and detail, it leaves only one class for smooth textures, namely, the class created for the cluster vectors on the origin of the histogram. The result is that during interpolation, softer gradient textures are rendered poorly, since they must be synthesized as the combination of "perfectly smooth" and "hard edge" textures. As the above discussion would suggest, we have observed a tradeoff in interpolation quality with respect to the value of p. By using a value of p like $\frac{1}{4}$, which falls between 0 and 1, we have achieved a cross between these two extreme cases.

2.2 Efficient Resolution Synthesis

In Sec. 2.1.2, we showed that the optimal interpolation is computed as a linear combination of the outputs of all M of the interpolating filters, with the weighting for the j–th filter output being the probability $p_{j|y}(j \mid y, \hat{\theta})$ that the image data is in class j. In most current implementations of the RS algorithm, the number M of classes is between 90 and 100. These numbers of classes are sufficient for the representation of a wide variety of behaviors like edges and smooth regions, which allows us to obtain very good interpolation results. However, since there are so many classes, the computation required for optimal interpolation may be excessive. Moreover, interpolating for classes which are different from the image data at hand is wasted. For example, to interpolate pixels which lie on a horizontal edge, there is little to gain from incorporating the filter designed for vertical edges. For these reasons, we have developed a modified interpolation algorithm, which employs a fast filter selection procedure in conjunction with the basic RS mechanism. We call this algorithm Efficient Resolution Synthesis (ERS). In ERS, instead of interpolating for every class, we use an average of only 1 or 2 filters. The computation required for ERS is much less than that for the "full" RS implementation. Also, while strictly speaking the resulting interpolation is sub-optimal, we have found that the results are nearly visually equivalent.

2.2.1 ERS interpolation

Interpolation by the ERS algorithm is locally adaptive, in the sense that a different set of filters may be used to interpolate every pixel. For a given pixel in the source image, we first execute a filter selection process to determine the set

$$j^* \subset \{1, \cdots, M\} , \tag{2.46}$$

whose elements are the indices of the classes for which we will interpolate. After this is done, then the ERS interpolation is computed as

$$\hat{X}_{\mathrm{ERS}} \;=\; \sum_{j\in j^*}(A_j Z+\beta_j)\frac{\exp\left(\frac{-1}{2\sigma^2}\|y-\mu_j\|^2\right)\pi_j}{\sum_{l\in j^*}\exp\left(\frac{-1}{2\sigma^2}\|y-\mu_l\|^2\right)\pi_l}\ . \tag{2.47}$$

The set j^* is characterized by the following filter selection criterion,

$$j\in j^* \text{ if } \|\mathrm{y}-\mu_{\mathrm{j}}\|^2 \le \|\mathrm{y}-\tilde{\mu}\|^2+2\left(\delta\sigma\right)^2\ , \tag{2.48}$$

where y is the cluster vector, σ^2 is the variance from the mixture density for cluster vectors, $\delta \ge 0$ is a threshold factor, and $\tilde{\mu}$ is the mean vector of the maximum-likelihood class $\tilde{\jmath}$:

$$\tilde{\jmath}=\arg\max_{1\le j\le M} p_{y|j}(y\mid j)\ . \tag{2.49}$$

Note that this framework ensures that j^* is never empty. Also, for large δ, all of the classes are included in j^*, which means that we can naturally characterize RS as the least selective form of ERS.

One statistic which directly affects interpolation time is the average number of filters used per input pixel. For a fixed value of δ, this statistic depends upon the image which is being interpolated; and it also depends upon the parameters of the mixture model for cluster vectors. Fig. 2.6 shows a plot of the average number of filters as a function of δ, for interpolation of one image by a factor of 4, by one implementation of ERS. As δ goes up from zero, this plot is monotonically increasing, from a minimum value of 1 to a maximum value of 100, which is the number of classes in this particular ERS implementation.

What is perhaps surprising is that for almost any value of δ, we have been able to obtain satisfactory interpolation results. This is even true for ERS with $\delta = 0$. For current implementations of ERS, we have been using a value of $\delta = 1$. We have found that this value is low enough to allow for reasonably fast interpolation, while allowing ERS to use a sufficient number of filters for satisfactory quality of interpolation.

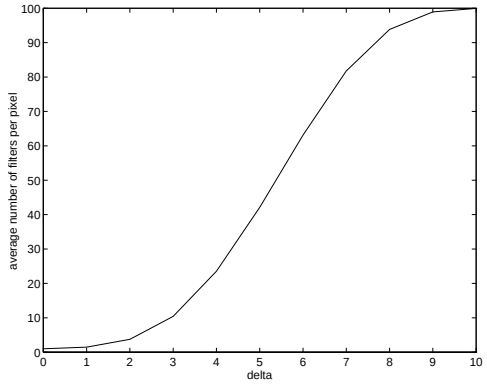


Fig. 2.6. Average number of filters per pixel in ERS, for varying values of δ. This plot was generated by interpolating the "cycles" image, by a factor of 4.

2.2.2 Fast filter selection for ERS

We have streamlined the basic ERS filter selection procedure, in order to make the filter selection even faster. The result is that we have decreased the time required for filter selection by a factor of around 5, while generally maintaining interpolation quality.

To select filters for interpolating a given pixel, the basic procedure is to obtain the cluster vector, and to use the filters whose classes have the highest likelihood. One problem is that to do this, we must compute the likelihoods for all of the available classes, and then compare them. To speed up the filter selection, it would be good if our list of candidate classes were limited to those which had some chance of satisfying the filter selection criterion.

The idea behind our approach is to find a simple partition of the space of cluster vectors, and to associate a limited list of candidate classes with each of the cells in this partition. These filter lists are obtained through a training procedure which we describe below. Since each of these filter lists will generally contain fewer than the total number of classes, the overall effect is to reduce the average candidate filter list length. In our experiments, we have used a partition with 1024 cells to decrease the average filter list length from 96 (the total number of classes) to just under 18.

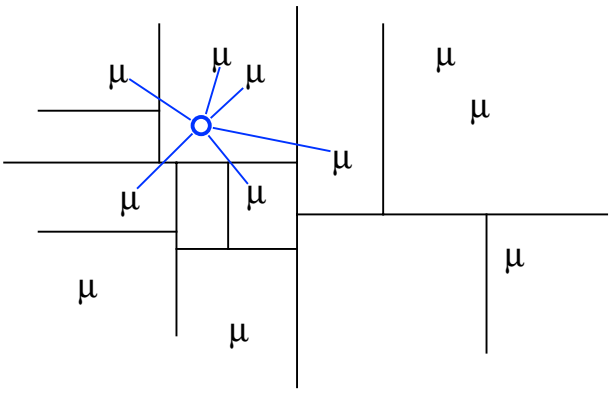


Fig. 2.7. Cluster vectors in the cell with the blue circle will only use filters from the set of candidates identified by the blue lines.

For a simple example, observe the scenario in Fig. 2.7. In this figure, we depict the means of the mixture density for cluster vectors with μ's. Now consider the task of filter selection for a cluster vector which lies inside the cell with the blue circle. It is clear that the only suitable candidates for this selection are connected to the blue circle with blue lines; the other classes are too far away to even be considered.

To define the partition, we use a binary tree. Each node in this tree corresponds to the comparison of some element of the cluster vector with a threshold; while each leaf in the tree corresponds to a cell in the partition. This means that to map an arbitrary cluster vector into its cell, we only have to execute a simple sequence of comparisons. This overhead computation turns out to be almost negligible.

To obtain the binary tree, we use the splitting technique known as the median cut algorithm, which was introduced by Heckbert [23]. Other splitting techniques [24, 25] may yield partitions which result in lower average filter list lengths. However, the median cut algorithm guarantees a symmetric binary tree, which has desirable computational consequences.

Once the partition has been defined, the next step is to execute a training procedure, in order to obtain a filter list for each of the cells in the partition. For this, we procure a large set of cluster vectors from example source images. Next, filters are selected for each of these cluster vectors, using the filter selection procedure for

ERS from Sec. 2.2. For each cell, we accumulate a histogram which summarizes the results of this training by keeping track of how many times each class was selected for cluster vectors in that cell. Then to construct the filter list for any cell, we only need to record the classes with nonzero entries in the corresponding histogram.

Intuitively, as the number of cells in the partition increases, the resulting average filter list length will decrease. One tradeoff which is related to this is that while a low average filter list length is desirable, the use of a large binary tree may require excessive memory. In addition, storage of a large binary tree may require excessive disk space.

2.3 Extensions to the RS Algorithm

In this section, we describe some extensions to the RS algorithm. One extension is a version of RS which may be used to interpolate color images. Another extension shows how we can build sharpening in to the RS interpolation procedure. Finally, we discuss how we handle interpolation of pixels on the image border.

2.3.1 Interpolation of color images

For RS to be generally useful, it should be applicable to the interpolation of color images. A color image consists of red, green, and blue components. In the context of color RS, the observation vector z consists of a triplet of observation vectors:

$$z = (z_{\mathrm{r}}, z_{\mathrm{g}}, z_{\mathrm{b}}) , \tag{2.50}$$

from the red, green, and blue components, respectively. We use the observation vector to estimate the vector

$$x = (x_{\mathrm{r}}, x_{\mathrm{g}}, x_{\mathrm{b}}) \tag{2.51}$$

of high-resolution pixels.

Ideally we should have three separate interpolators, one for computing each of $x_{\mathrm{r}}, x_{\mathrm{g}}$, and x_{b}. Each of these interpolators should be a function of all three of the

observation vectors $z_{\rm r}, z_{\rm g}$, and $z_{\rm b}$. However, we have found that very good color interpolation can be achieved by using an interpolator designed for monochrome images. We apply the same interpolating filters in the same proportions to each of the red, green, and blue components. The filters and their proportions are chosen on the basis of the image luminance component.

We first compute a cluster vector $\overline{y}$ as

$$\overline{y} \;=\; f(\overline{z}) \;, \tag{2.52}$$

with the projection operator $f(\cdot)$ from Sec. 2.1.4, and with

$$\overline{z} = 0.299 z_{\rm r} + 0.587 z_{\rm g} + 0.114 z_{\rm b} \;. \tag{2.53}$$

Using this cluster vector we choose the set j^*, through the selection criterion of (2.48), using $\overline{y}$ in place of y. Next, we interpolate for each of the red, green, and blue components separately. For the red component,

$$\hat{X}_{\rm r} \;=\; \sum_{j \in j^*} (A_j Z_{\rm r} + \beta_j) \frac{\exp\left(\frac{-1}{2\sigma^2}\|\overline{y} - \mu_j\|^2\right)\pi_j}{\sum_{l \in j^*} \exp\left(\frac{-1}{2\sigma^2}\|\overline{y} - \mu_l\|^2\right)\pi_l} \;, \tag{2.54}$$

where $Z_{\rm r}$ is the observation vector obtained from the red plane. The green and blue components are processed similarly.

We have used this approach to obtain good interpolation results with a variety of color images. In addition, there are a number of computational benefits. Since we use the same cluster vector $\overline{y}$ for each of the components, only one execution of the filter selection procedure is required for each pixel in the source image. This benefit is significant since much of the computation required for ERS is needed for filter selection. Also, we only need to store a single interpolator, as opposed to one for each of the red, green, and blue components.

2.3.2 Built-in sharpening

Regardless of the interpolation method which is used, interpolated images can often appear soft or smooth. For this reason, the appearance of interpolated images can often be improved by sharpening. With most interpolation methods, this

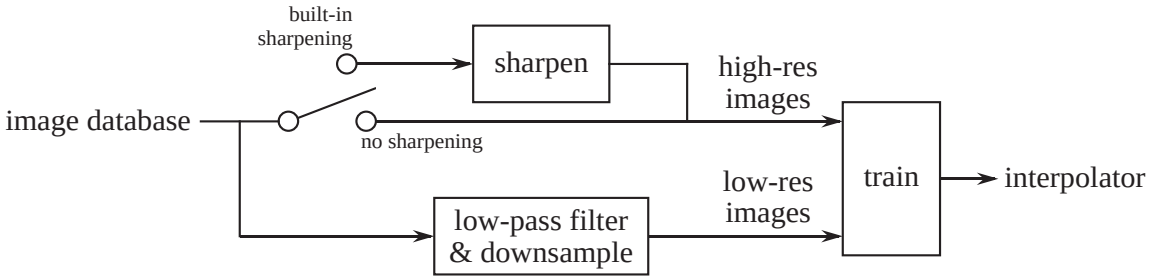


Fig. 2.8. Interpolator design with and without built-in sharpening.

sharpening must be performed after interpolation is complete. Unfortunately, such post-processing can also enhance noise along with detail. However, with RS, we can design an interpolator which has a built-in sharpening property, so that sharpening and interpolation are simultaneously performed in a minimum mean-squared error sense.

The important observation here is that the interpolator design amounts to the specification of a stochastic model, which relates low-resolution image data to high-resolution image data. This specification is completely guided by the training set. Therefore we would expect that interpolated images can only be as sharp as the high-resolution renderings in the training set.

Because of the above observation, the procedures for making interpolators with and without built-in sharpening are quite similar. The only difference lies in how the respective training sets are generated. This fact is illustrated in Fig. 2.8. We first procure a database of images, and we regard these images as "high-resolution" renderings. We decimate them to create their corresponding "low-resolution" images. If built-in sharpening is desired, then the next step is to sharpen the high-resolution images using an unsharp mask.

The rest of the interpolator design procedure is the same, regardless of whether sharpening is desired. Therefore the interpolator with sharpening uses the same interpolation algorithm, but with a different parameter estimate for ψ.

2.3.3 Interpolation of border pixels

On image borders, there are pixels which are not centers of 5×5–pixel windows. To interpolate these pixels, it is necessary to extrapolate the values of some neighboring pixels. To do this, we simply use the value of the nearest known pixel. This approach generally gives results which are satisfactory, at no computational cost. We have tried more sophisticated approaches, which do a slightly better job than our "nearest neighbor" approach. However, this is at the cost of increased computation.

2.4 Results

In Fig. 2.9, we present interpolation results for a factor of $L = 4$, for the following methods: pixel replication; bilinear interpolation; ERS; and ERS with built-in sharpening. In other comparisons, we have found that ERS compares very favorably with different interpolation methods; we have included pixel replication and bilinear interpolation for this comparison because they are common techniques.

Each of the methods represented in Fig. 2.9 seems to perform adequately for interpolating smooth textures. However, we feel that this comparison shows that ERS renders edges and detail relatively continuous and sharp. The reason for this is that ERS uses several filters, each of which is optimal for interpolating a single type of behavior; while each of the competing methods uses only one interpolating filter.

Note that we have used ERS, as opposed to RS, for Fig. 2.9. For a comparison of the results of ERS with those of RS, we have included Fig. 2.10. Both these images were interpolated using the same filter parameters. The difference between them is that to compute the image of Fig. 2.10, all 99 interpolating filters were used per input pixel, while to compute image (e) of Fig. 2.9, an average of only 1.5 filters were used. This example is typical: ERS and RS results are usually visually equivalent, while ERS involves significantly less computation (assuming that an appropriate value of δ has been used for ERS).

In this report, we have emphasized the importance of having distinct classes of image data for important features like edges. To encourage the formation of such classes, we use the projection operator $f(\cdot)$, which is described in Sec. 2.1.4. This operator includes a parameter p, which critically affects the types of classes that are established during clustering. To demonstrate the control of this parameter, we have included Fig. 2.11, which should be compared with image (e) of Fig. 2.9. For Fig. 2.11, we have used a value of $p = 1$ for the projection operator; while for image (e) of Fig. 2.9, we have used a value of $p = \frac{1}{4}$. Using a value of $p = \frac{1}{4}$ results in classes for more visually different types of behaviors like edges and texture; this makes interpolation more accurate. Note the difference in edge rendition, for example, for the number ("21") on the helmet in the upper-right corner.

2.5 RS for Specific Types of Images

At first, it may seem like the fact that the RS interpolator parameters depend upon the training images is a disadvantage. However, in this section we demonstrate that this is actually an advantage, by showing that RS can be trained to interpolate specific types of images while ameliorating some of their characteristic artifacts. Examples of such images are JPEG-compressed images, which exhibit the well-known blocking and ringing artifacts, and digital camera images, which exhibit artifacts of the acquisition, compression, and other processing.

Our approach is to construct the set of training images in such a way that RS "learns" to estimate sharp image data, free of artifacts like those described above, from observations of image data which contain those artifacts. Note that this is only a modification of how we construct the training set. The remaining procedures of the interpolator design and the interpolation itself, are exactly the same as they have been described previously. In the rest of this section, we will see that this simple approach yields good results for the cases of JPEG and digital camera images.

(a) Source image.

(b) Pixel replication.

(c) Bilinear interpolation.

(d) ERS.

(e) ERS with built-in sharpening.

Fig. 2.9. Results of interpolation by a factor of $L = 4$.

(a) Efficient Resolution Synthesis.

(b) Resolution Synthesis.

Fig. 2.10. Comparison of ERS and RS interpolations, by a factor of $L = 4$ (both with built-in sharpening).

(a) ERS with $p = \frac{1}{4}$.

(b) ERS with $p = 1$.

Fig. 2.11. Comparison of results of ERS interpolation by a factor of $L = 4$, with built-in sharpening, using values of $\frac{1}{4}$ and 1 for the variable p in the projection operator $f(\cdot)$ for cluster vectors.

2.5.1 RS for JPEG images

To generate interpolator parameters for interpolating JPEG images by a factor of $L = 2$, we build the training set as illustrated in Fig. 2.12. We obtain a set of original images which are free of JPEG artifacts and sharpen them to create the high-resolution training images. Next, we generate low-resolution images from the original images by low-pass filtering followed by downsampling by a factor of 2, and we compress them at a specified quality level in order to create the low-resolution training images. The JPEG quality level is an integer between 0 and 100 which determines the compression rate; low quality levels give more compression along with more objectionable compression artifacts, and *vice versa* for high quality levels. This is an important factor since ultimately it dictates the level of artifact suppression that will be imposed during the interpolation process. We generated three sets of interpolator parameters, one for each of low-, medium-, and high-quality factors 25, 50, and 75. For interpolation of an arbitrary JPEG image, we use the parameters designed for the closest quality level.

In Fig. 2.13, we show interpolation results for JPEG quality level 25. Using this quality level tends to result in fairly significant artifacts. Note that if the interpolator parameters are optimized for interpolating JPEG images at this quality level, the blocking and ringing artifacts are significantly reduced while other edges in the image are rendered sharper and more continuous. On the other hand, if we use RS interpolator parameters which are not designed for JPEG images, the block boundaries can actually be enhanced. This is because if it is not trained to interpolate JPEG images compressed at this quality level, RS tends to treat the block boundaries like edges.

2.5.2 RS for digital camera images

The process of generating the training set for designing an RS interpolator for digital camera images is more challenging. This is because for the high-resolution training images, we need "ideal" digital camera images. These images should exhibit the aspects of driver processing which improve image appearance like sharpening and

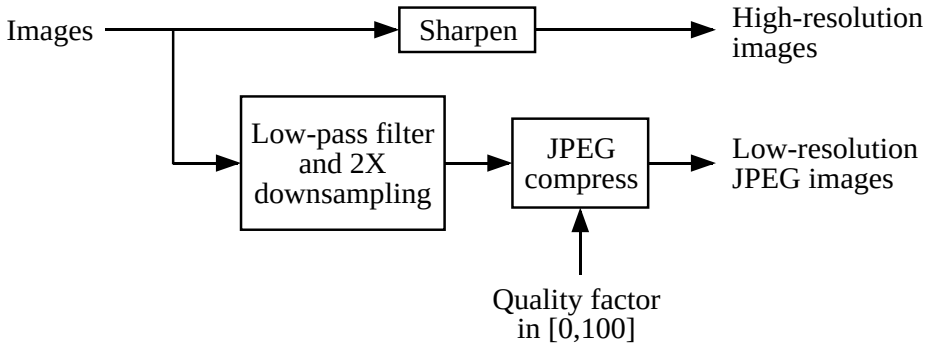


Fig. 2.12. Construction of training set for RS interpolation of JPEG images by a factor of 2.

color transformations, but they should not exhibit objectionable artifacts like those from the acquisition and compression processes. This means that we cannot expect to obtain high-quality original training images directly from a digital camera. We were able to get around this, and to design an RS interpolator for images acquired by one particular model of digital camera, by using a digital camera simulator.

An overview of the device simulation process appears in Fig. 2.14. Note that the image at the output of the simulator is at half the resolution of the input image. The simulator contains components for various aspects of the camera processing such as lens blur; image acquisition by camera sensors; sensor noise; camera driver processing like sharpening and color transformations which increase saturation; and compression. One input which the simulator takes is a quality setting which controls the level of compression. We designed three RS interpolators for interpolation by a factor of $L = 2$, one for images acquired using each of the three quality settings.

The process by which we generate a pair of training images is illustrated in Fig. 2.15. The objective here is to create one "ideal" digital camera image at a base resolution, and one typical digital camera image at half the base resolution, for a given quality setting. We use high-quality images from a photo CD as inputs to this process. One convenient aspect of this is that the file format of the photo CDs actually provides the images at various resolutions.

(a) Pixel replication.

(b) RS optimized for JPEG images compressed at quality level 25/100.

(c) RS not optimized for JPEG images.

Fig. 2.13. Results of 2X interpolation of a JPEG image compressed at quality factor 25/100.

To generate the low-resolution training images for a particular camera quality setting, we simply submit the base-resolution renderings of the photo CD images to the camera simulator.

We use the process illustrated in Fig. 2.16 to generate the "ideal" high-resolution training images. Our approach is to run the camera simulation using a very high-resolution image, and then to low-pass filter and downsample the simulator output in order to average out (or at least ameliorate) compression artifacts. The input here is the photo CD image at four times the base resolution. We first interpolate the input image by a factor of 2 using a "standard" RS interpolator, *i.e.*, one designed for images that are free of artifacts. We submit the interpolated image to the camera simulator with the highest quality setting to avoid compression artifacts, and the next step is to low-pass filter and downsample the output image by a factor of 4. This yields an image at the base resolution.

The last step in this process is to sharpen the base-resolution training image so that its appearance is even sharper than that of the corresponding low-resolution training image. This requires quite a bit of sharpening, since the low-resolution training image has already been sharpened by the camera driver processing. We used unsharp masking with a sharpening factor of 4.0. Before applying the unsharp mask, we applied a very light blur to the image, since sharpening of this magnitude would otherwise seriously degrade the image by enhancing spurious noise. The impulse response of the blurring filter we used is

$$\begin{matrix} & 0.12 & \\ 0.12 & 0.52 & 0.12 \\ & 0.12 & \end{matrix},$$

with the origin corresponding to coefficient 0.52.

Finally, we ensure that the pixels in the low- and high-resolution training images are properly registered relative to one another. Our approach is to shift the pixels in the high-resolution images by a spatial offset so that, for example, the image data at location (i, j) in the low-resolution image corresponds to the image data in the

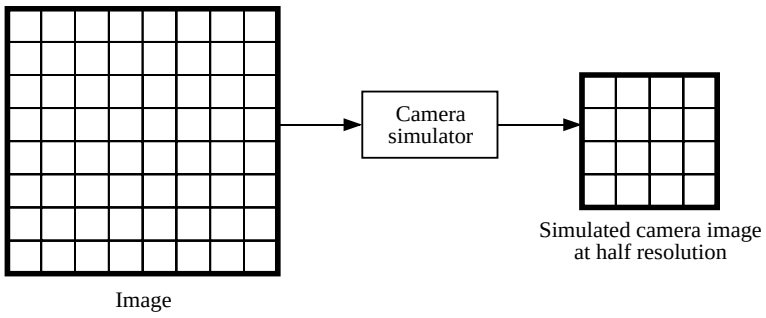


Fig. 2.14. Overview of the digital camera simulator.

$L \times L$ block of the high-resolution image whose upper, left-hand corner has indices (Li, Lj). We computed the required offset as the maximizer of the sample correlation between the high-resolution image and a pixel-replicated version of the low-resolution image (with the pixel replication being by a factor of 2). We found this offset to be consistent over several images, across the three quality modes of the camera, to be 2 pixels down in the vertical direction, and 1 pixel to the right in the horizontal direction. Note that the offset for registering training images created using a different digital camera simulator would probably be different.

We present experimental results in Fig. 2.17. The input image is an actual digital camera image, taken using the lowest-quality setting. (We used the model of camera which the camera simulator was designed to simulate.) Three artifacts which the original image exhibits in particular are the thick, "jaggie" edges typical of digital camera images; chromatic aberrations; and compression artifacts. Note that the result interpolated by RS optimized for images acquired in the lowest-quality setting exhibits fewer blocking and chromatic artifacts, while many of the strong edges in the scene are sharper and more continuous.

2.6 Conclusions

In this chapter, we have introduced the interpolation algorithm known as Resolution Synthesis. We have provided an analytical derivation of the RS algorithm, and we have established that under certain assumptions, RS actually generates a MMSE

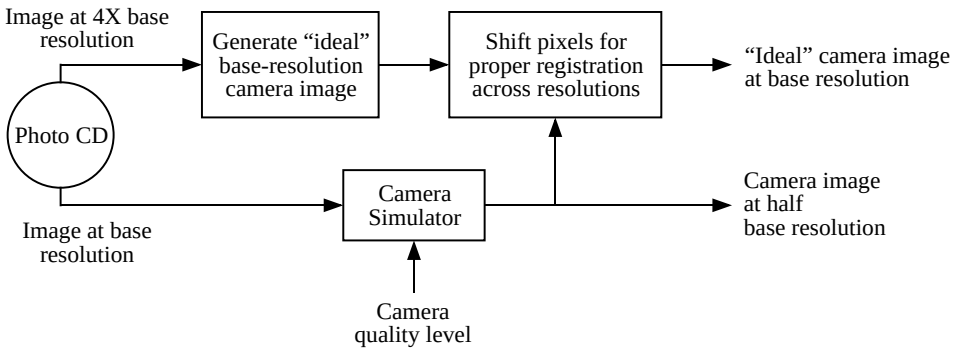


Fig. 2.15. Construction of a training image pair for RS interpolation of digital camera images by a factor of 2.

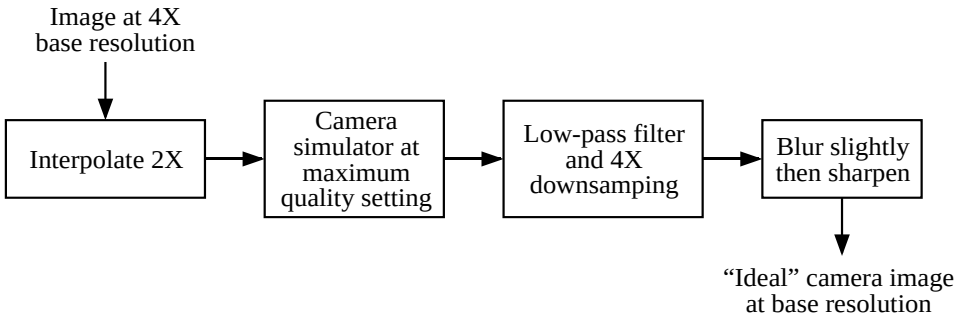


Fig. 2.16. Method for generating an "ideal" high-resolution digital camera image using the digital camera simulator.

(a) Pixel replication.

(b) RS optimized for digital camera images acquired in normal mode.

Fig. 2.17. Results of 2X interpolation of a digital camera image acquired in "normal" (lowest-quality) mode.

estimate of the target-resolution image, given the source image. It should be emphasized here that RS provides optimal image interpolation in the form of a closed-form computation. This is in contrast to the method of Schultz and Stevenson [11], which requires an expensive iterative optimization. It is also worth noting that the model for image data which is incorporated by RS has a somewhat general form. This may make it attractive for various other applications in image processing. For example, this model may be used as the basis for an image coding scheme which incorporates an arithmetic coder, similar to that implemented by Popat and Picard [12]. Finally, we have presented results which demonstrate that both RS and ERS yield interpolations of very satisfactory subjective quality.

3. RS-BASED JPEG IMAGE COMPRESSION

In the preceding chapter, we demonstrated that by using JPEG-decompressed images as the low-resolution images in the training set, we can train RS to perform high-quality interpolation of JPEG images. In this chapter, we introduce a new interpolative image coding scheme which exploits this aspect of RS. The basic idea here is that as an alternative to JPEG-encoding an image using a low quality setting, we can JPEG-encode a low-resolution rendering of the image at a higher quality setting; then at the decoder, we can use RS to recover the full-resolution rendering. We compare the RS-based scheme to JPEG used alone, and we demonstrate that for lower bit rates (at or below roughly 0.4 bits per pixel), the RS-based scheme enables us to achieve better subjective image quality (for approximately equal bit rates), as well as an improved rate-distortion tradeoff.

The idea of explicitly incorporating the interpolation process into data compression is not new. A general description of one such scheme is given in [26], and applications to image compression in particular are described in [27], [28], and [29].

Our approach is illustrated in Fig. 3.1. The first step is to perform 2×2 block-averaging on the input image, which yields the low-resolution rendering. Next, we JPEG-encode the low-resolution image. Note that the image which is JPEG-coded is smaller, and that aside from the JPEG overhead, the present scheme does not require any additional side information to be encoded along with the image. This means that we can afford to perform the JPEG coding at a higher quality setting. At the decoder, we first perform JPEG decompression, and then apply RS (for interpolation by a factor of 2) to the decompressed image. An important fact here is that the RS

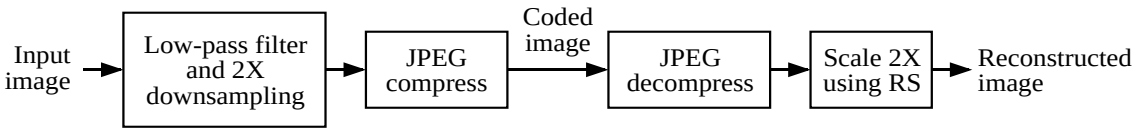


Fig. 3.1. Overview of RS-based JPEG image compression.

interpolator parameters must have been trained for interpolating images compressed at the same or a similar quality setting as that which was used at the encoder.

In Fig. 3.2, we present two rate-distortion curves for compression of a sample image, with one curve obtained using RS-based JPEG compression, and with the other obtained using JPEG compression alone. For our distortion metric, we have used the average of the mean-squared errors for each of the red, green, and blue planes (for pixel values in $[0, 1]$, after removing an assumed gamma correction). We observe that for lower bit-rates, the RS-based scheme outperforms JPEG alone. Finally, in Fig. 3.3, we present images cropped out of two of the images represented in Fig. 3.2, rendered at 100 dots per inch. Note that while the image compressed using the RS-based method exhibits fewer blocking and chromatic artifacts, it was compressed at a slightly lower bit-rate.

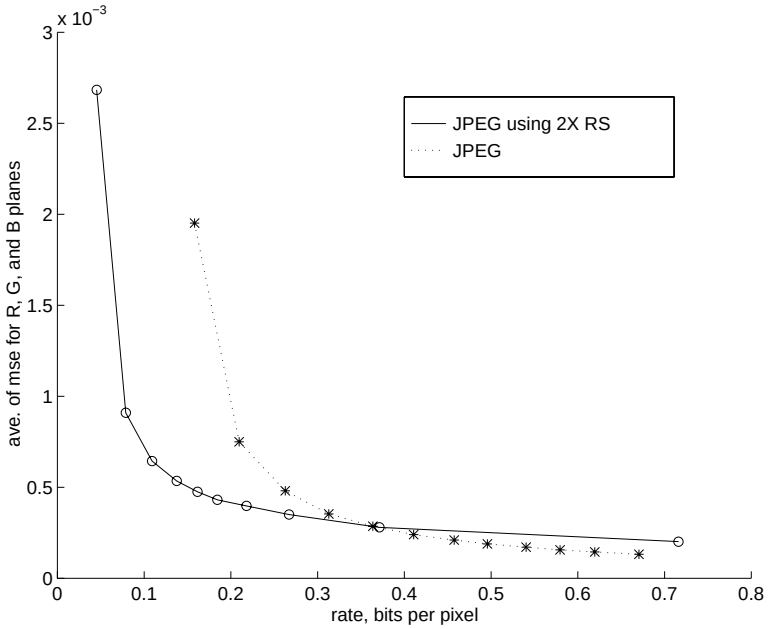


Fig. 3.2. Rate-distortion characteristics for RS-based JPEG compression and for JPEG compression alone, for the image shown in Fig. 3.3.

(a) RS-based JPEG compression, 0.22 bits per pixel.

(b) JPEG compression alone, 0.26 bits per pixel.

Fig. 3.3. Image compression results.

4. TREE-BASED IMAGE INTERPOLATION

In this chapter we introduce two new image interpolation techniques. One is called Tree-based Resolution Synthesis (TRS). TRS is similar to RS in that it works by first performing a local classification, and then applying a linear filter which is optimal for the selected class. However, it is different in that the classification is obtained using a binary tree. The second interpolation method, which we call Enhanced Tree-based Resolution Synthesis, works by first generating the TRS interpolation and then processing the interpolated image to correct for interpolation artifacts and to impart additional enhancements.

4.1 Introduction

A digital image is usually only available at one or a few fixed resolutions, so that in order to render it at a higher resolution, some image interpolation technique must be applied. This involves some inference on the part of the interpolation algorithm; and for most images, the result is that the higher-resolution image exhibits some objectionable artifact of the interpolation process. In answer to this, we present Enhanced Tree-based Resolution Synthesis (ETRS), a two-stage image interpolation algorithm which first performs an interpolation, and second processes the interpolated image to correct interpolation artifacts and to impart additional enhancements.

We believe that this work represents two important contributions to the field of image interpolation. One contribution is that for the first stage of ETRS, we introduce a new approach to minimum mean-squared error (MMSE) image interpolation called Tree-based Resolution Synthesis (TRS). TRS works by first performing a fast local classification of a window around the pixel being interpolated, and then by applying

an interpolation filter designed for the selected class. The idea behind TRS is to use a regression tree as a piecewise linear approximation to the conditional mean estimator of the high-resolution image given the low-resolution image. Intuitively, having the different regions of linear operation allows for separate filtering of distinct behaviors like edges of different orientation and smoother gradient transitions.

Another contribution is that we use the interpolated image from the first stage of ETRS to compute an enhanced interpolated image having sharper appearance and fewer interpolation artifacts. This is the second stage of ETRS. To compute a pixel in the enhanced interpolated image, we take the corresponding pixel in the interpolated image and add an adjustment to it. The adjustment is computed using a simple formula designed to impart stronger adjustments around edges and in texture regions, where blurriness and other artifacts are more likely to be manifested. The inputs to this formula are local estimates of the sign and the magnitude of the interpolation error, which we obtain using a second regression tree. This process is recursive, since the input to the regression tree consists of pixels in the interpolated image as well as pixels in the enhanced interpolated image, which have previously been adjusted.

Before ETRS interpolation can be executed, we must already have generated the regression trees for the two stages by training on sample images. These training procedures are computationally demanding, but they only need to be performed once. We will see that the resulting predictors may be used effectively on input images which were not used in the training.

The steps involved in the training procedures for the two stages are quite similar. In each procedure, we first grow a large regression tree using data from the training images. For this, we start with a tree having only one leaf node, and we "grow" it one leaf node at a time, splitting that leaf node at each step which yields the greatest decrease of a relevant cost function. Next we prune the tree back using a separate training set. The idea here is to throw out extraneous branches, leaving behind a tree structure which efficiently represents a wide range of behaviors observable in the training images. As a final step, we generate new filters for the terminal nodes of

the regression tree using a very large set of training data, while leaving the classifier (or the partition defined by the binary tree) unchanged. Since the resulting filters are generated from much larger training sets, they tend to be more stable and this improves performance.

We will see that this work represents a departure from traditional spline-based interpolation techniques [3, 2, 4, 1] which usually rely on some assumption about the continuous structure of an underlying interpolating function. This can result in blurring and ringing artifacts which can be exacerbated by less sophisticated post-processing fix-ups such as sharpening by unsharp masking. This work is also different from [11], which is based on a different underlying stochastic model. However, it does share some similarity with the wavelet-based approach described in [30] as well as the other edge-directed techniques described in [10, 9, 8, 7, 6], in the sense that the purpose of the enhancement stage in ETRS is to locate edge regions and to treat them specially for sharper appearance.

In the remainder of this report, we give a detailed description of the ETRS algorithm. First, in Sec. 4.2, we describe TRS interpolation, and we develop the training method which we use to obtain the interpolator parameters. Next, in Sec. 4.3, we describe ETRS interpolation, and we describe the training method used to generate the predictor for estimating the conditional distribution of the interpolation error. Then in Secs. 4.4 and 4.5, we present experimental results and conclusions.

4.2 Tree-based Resolution Synthesis

An overview of the TRS algorithm appears in Fig. 4.1. We begin by training on sample images to generate the interpolator parameters. This is a computationally expensive procedure, but it only needs to be executed once. The resulting interpolator parameters are stored and may be used to interpolate images from outside the training set. In fact, the results which we will show in Sec. 4.4 were generated using images from outside the training set. In this section, we will first describe the TRS interpolation procedure, assuming for the time being that the interpolator parameters

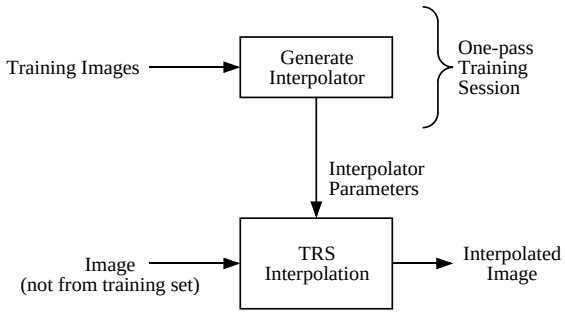


Fig. 4.1. Overview of the TRS algorithm.

are known. Then we will describe the training process which we use to obtain those parameters.

In this section, when we refer to images, we mean monochrome images with pixel values in the range $[0, 255]$. To interpolate color images, we apply the interpolation procedure described in Sec. 4.2.1 to each of the red, green, and blue planes separately.

4.2.1 TRS interpolation

In Tree-based Resolution Synthesis (TRS), we generate an $L \times L$ block of high-resolution pixels for every pixel in the low-resolution source image as illustrated in Fig. 4.2. We do this by filtering the corresponding $W \times W$ window of pixels in the low-resolution image, with the filter coefficients selected based on a classification. Thinking of the desired high-resolution pixels as the realization of an L^2-dimensional random vector X and the corresponding low-resolution pixels as the realization of a W^2-dimensional random vector Z, our approach in TRS is to use a regression tree which approximates the conditional mean estimator of $(X|Z)$, so that the vector $\hat{X}$ of interpolated pixels satisfies

$$\hat{X} \approx \mathrm{E}\,[X|Z] \ . \tag{4.1}$$

It is well-known that the conditional mean estimator minimizes the expected mean-squared error [13]. (Note that we will use capital letters to represent random quantities, and lowercase letters for their realizations.)

A closed-form expression for the true conditional mean estimator would be difficult to obtain for the present context. However, the regression tree T which we use provides a convenient and flexible piecewise linear approximation, with the M different linear regions being polygonal subsets which comprise a partition of the sample space $\mathcal{Z}$ of low-resolution vectors Z. We will associate a unique integer index in $\{0, ..., M-1\}$ with each of these subsets, and we will refer to them as classes. In fact we will see that these classes correspond to visually distinct behaviors like edges of different orientation. An important consequence of this is that our approach is less likely to perform excessive averaging across edge boundaries, since each linear filter in T is optimized for its respective class. Another advantage inherent in the present approach is that the regression tree generates a hierarchical partition of $\mathcal{Z}$, making the classification process very efficient.

With the main ideas in place, we return to Fig. 4.2 for a better look. To interpolate the shaded pixel in the low-resolution image, we first procure the vector z by stacking the pixels in the 5×5 window centered there. Then we obtain interpolated pixels as

$$\hat{x} = A_j z + \beta_j \ , \tag{4.2}$$

where A_j and β_j are respectively the $L^2 \times W^2$ matrix and L^2-dimensional vector comprising the interpolation filter for class j, and j is the index of the class of image data obtained as

$$j = C_T(z) \ , \tag{4.3}$$

where $C_T(\cdot) : \mathcal{Z} \to \{0, ..., M-1\}$ is a function which embodies the classifying action of T. To evaluate $C_T(z)$, we begin at the top and traverse down the tree T as illustrated in Fig. 4.3, making a decision to go right or left at each nonterminal node (circle), and taking the index j of the terminal node (square) which z lands in. Each decision has the following form,

$$e_m^t \left(z - \mu_m\right) \begin{array}{c} > \\ < \end{array} 0 \ , \tag{4.4}$$

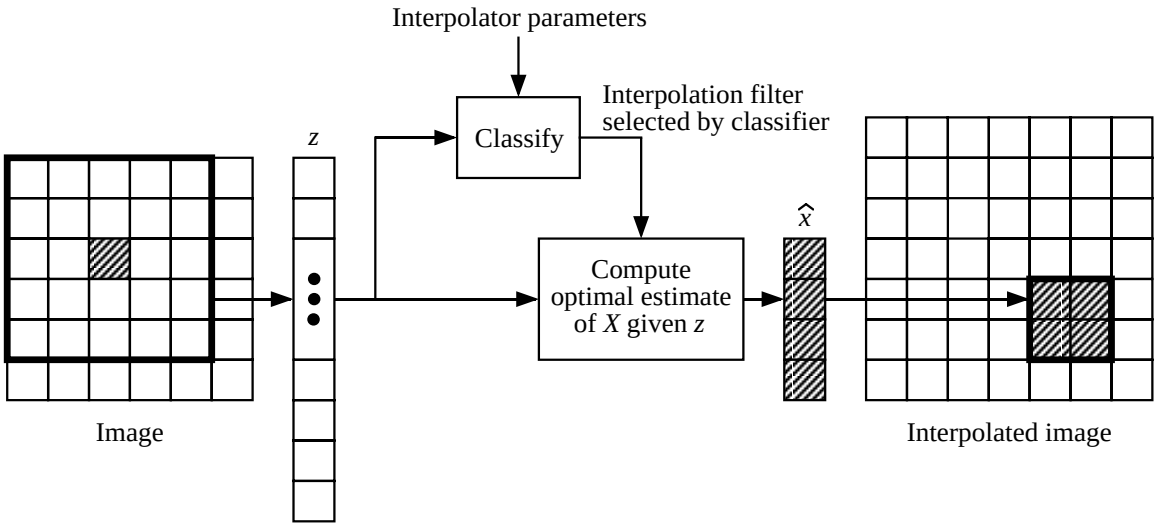


Fig. 4.2. TRS image interpolation by a factor of $L = 2$.

where m is the index of the node, e_m and μ_m are W^2-dimensional vectors, and a superscript "t" denotes taking the transpose. Note that (4.4) decides whether z is on one side of a hyperplane or the other, with μ_m being a point in the hyperplane and with e_m specifying its orientation. By convention, we go left if the quantity on the left-hand side of (4.4) is negative, and we go right otherwise.

4.2.2 Generating parameters for the TRS interpolator

In the preceding section, we demonstrated the application of a regression tree T to the problem of optimal image interpolation, under the assumption that we had the parameters for T. In practice, we precompute those parameters in a training process which we describe in this section. Specifically, our objective here is to obtain numerical values for the integer number $M \geq 1$ of terminal nodes in the tree; the decision rules

$$\{(e_m, \mu_m)\}_{m=0}^{M-2} \tag{4.5}$$

for the nonterminal nodes (assuming that $M > 1$); and the interpolation filters

$$\{(A_m, \beta_m)\}_{m=0}^{M-1} \tag{4.6}$$

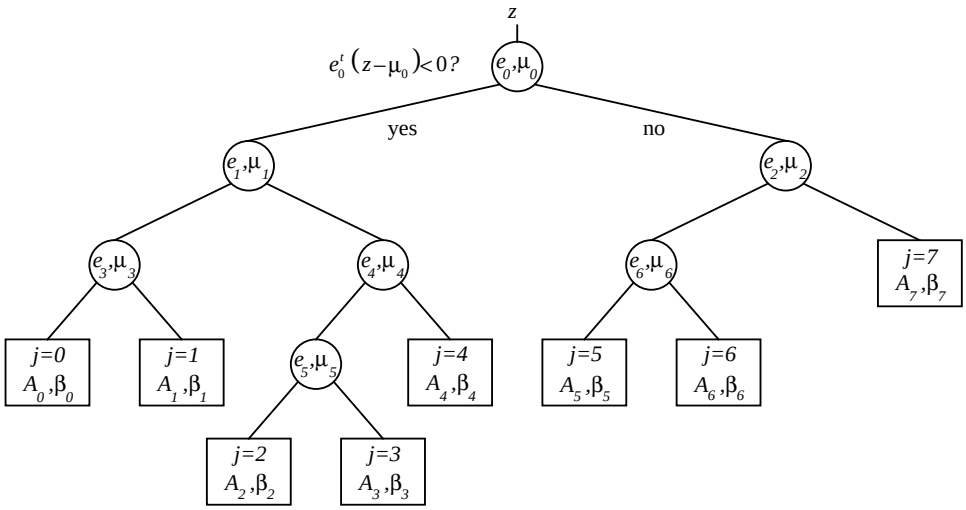


Fig. 4.3. Binary tree structure used in TRS.

for the terminal nodes. To compute these parameters, we use training sets consisting of interpolation training vector pairs, which we assume are independent realizations of (X, Z). An interpolation training vector pair is extracted from low- and high-resolution renderings of the same image as illustrated in Fig. 4.4. We will describe how we generate these paired renderings of training images later.

Our procedure is based on that given by Gelfand, Ravishankar, and Delp, in [31], suitably modified for the design of a regression tree rather than a classification tree. We first use one training set G to grow the tree, and then we use a different training set P to prune it back. Finally we generate new interpolation filters for the terminal nodes using a very large training set D. We take the tree resulting from the filter generation procedure as the final interpolator T. This process is illustrated in Fig. 4.5.

An important difference between our procedure and that of Gelfand, Ravishankar, and Delp, is that our procedure only involves one growing phase followed by one pruning phase. We have found that this is enough to generate a tree structure which efficiently represents a variety of classes of image data. In the procedure of [31], the growing-then-pruning continues in cycles, with the roles of the sets G and P (as "growing" and "pruning" sets) alternating between cycles. Interestingly they show that the sequence of classification trees remaining after each pruning phase is nondecreasing, and actually converges after two successive iterations have yielded the same classification tree. The objective of these iterations is based on the observation made by Breiman, Friedman, Olshen, and Stone, that the classification tree should neither overfit nor underfit the training data [32]. Their observation is no less relevant in our context; however, we have found that the filter generation process at the end of the training procedure renders performance of the interpolator much less sensitive to the number of classes in the tree.

In the rest of this section, we describe the tree growing procedure in Sec. 4.2.2.1, the pruning procedure in Sec. 4.2.2.2, and the final filter generation procedure in Sec.

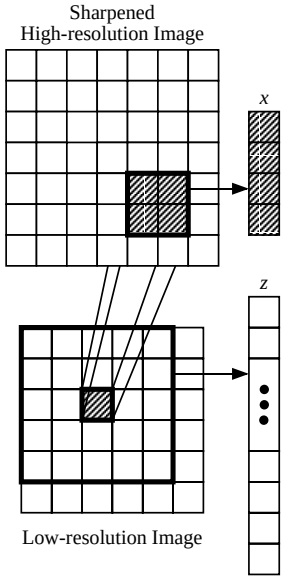


Fig. 4.4. An interpolation training vector pair (x, z).

4.2.2.3. Last, in Sec. 4.2.2.4, we describe how we generate the training images and how we procure the training sets G, P, and D.

We will refer to a tree-structured interpolator with ℓ terminal nodes as $T(\ell)$. To evaluate the quality of $T(\ell)$, we will compute its sample mean-squared interpolation error, which depends considerably upon the particular set of interpolation training vector pairs used in the computation. During the growing phase we will use set G, and we will use the notation $\mathcal{E}_G(T(\ell))$. Note that this is a resubstitution estimate, and should not be regarded as an honest estimate the true mean-squared error of $T(\ell)$ [32]. However, we will only use $\mathcal{E}_G(\cdot)$ to compare trees generated during the growing phase. We will similarly only use $\mathcal{E}_P(\cdot)$, computed relative to the pruning set P, to compare trees generated during the pruning phase.

4.2.2.1 Growing the tree-structured interpolator

During the tree growing phase, we generate an increasing sequence of trees

$$T(1), T(2), ..., T(M_{\max}) \ , \tag{4.7}$$

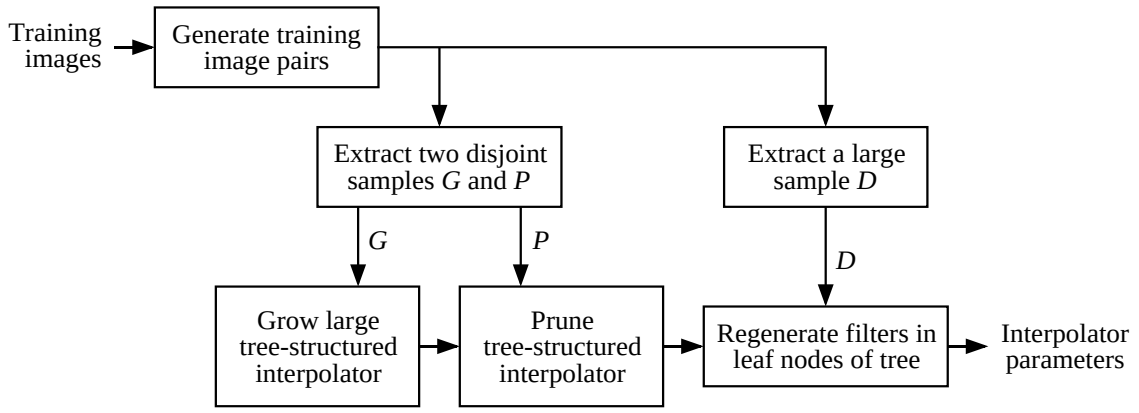


Fig. 4.5. Overview of the TRS training process.

where $M_{\max}$ is a user-specified maximum number of terminal nodes. The first tree $T(1)$ consists of only one terminal node, and each successive tree is is generated by splitting one of the terminal nodes in the preceding tree. An important fact is that we select $M_{\max}$ to be large enough so that $T(M_{\max})$ actually overfits the set G. We do this so that after extraneous branches are pruned away from $T(M_{\max})$, the tree-structured interpolator which is left over will have classes which represent the entire range of different behaviors present in the training images. In our experiments (using data sets procured as we describe in Sec. 4.2.2.4), we have obtained reasonable interpolation results by setting $M_{\max}$ to 500. This tends to yield final tree sizes (*i.e.*, after the pruning phase is complete) of between roughly 20 and 50 classes.

To continue our description of the tree growing procedure, we generate $T(\ell+1)$ from $T(\ell)$ (for $\ell \geq 1$) by splitting that terminal node with index m^* which yields the greatest reduction in the sample mean-squared error. More specifically, we split the terminal node m^* of $T(\ell)$ selected as

$$m^* = \arg \min_{0 \leq m < \ell} \mathcal{E}_G(T'_m(\ell+1)) \ , \tag{4.8}$$

where $T'_m(\ell+1)$ is the candidate tree with $(\ell+1)$ terminal nodes, generated by splitting node m in $T(\ell)$. Note here that we will use primes with variables associated with candidate trees.

To generate the candidate tree $T'_m(\ell+1)$ from $T(\ell)$, we do two things. First, we use the subset of G belonging to class m, defined as

$$G_m = \Big\{(x,z) \in G : C_{T(\ell)}(z) = m\Big\} \ , \tag{4.9}$$

to generate the candidate decision rule

$$(e'_{\ell-1}, \mu'_{\ell-1}) \tag{4.10}$$

for the candidate nonterminal node in $T'_m(\ell+1)$. Note that we use the subscript $(\ell-1)$ since this will be the index of the new nonterminal node in $T(\ell+1)$. Second, we use (4.10) to divide G_m into two sets G'_m and G'_ℓ, and we generate new interpolation filters

$$(\tilde{A}'_m, \tilde{\beta}'_m) \tag{4.11}$$

and

$$(\tilde{A}'_\ell, \tilde{\beta}'_\ell) \tag{4.12}$$

for the two candidate terminal nodes in $T'_m(\ell+1)$. Here, we use the indices m and ℓ since they will be the indices of the two new terminal nodes in $T(\ell+1)$. Note also that we will use superscript tildes to denote interpolation filters generated during the growing stage of the training.

Splitting the m–th terminal node in $T(\ell)$. By generating the decision rule (4.10) we split the polygonal subset $\mathcal{Z}_m$ of $\mathcal{Z}$ belonging to class m defined as

$$\mathcal{Z}_m = \Big\{z \in \mathcal{Z} : C_{T(\ell)}(z) = m\Big\} \ . \tag{4.13}$$

It is common for a split to have the form of a hyperplane in $\mathcal{Z}$, but the form which we will use is actually quite general since we make no restriction on the orientation of the hyperplane. Note that in this section, we restrict our attention to the m-th terminal node of $T(\ell)$. To keep the notation simple, we will be using the notations X and Z to refer to the random variables $(X|Z \in \mathcal{Z}_m)$ and $(Z|Z \in \mathcal{Z}_m)$.

To decide on a procedure for specifying the decision rule, we consider the following objectives. First, we want to divide the classes relative to the space of output vectors since this relates directly to our objective of reducing mean-squared interpolation error. But second, the split which we define must generate a hyperplane in $\mathcal{Z}$ that separates the realizations of Z in G_m.

To achieve these objectives, we define the split based on a vector $\bar{X}$, defined as

$$\bar{X} = B\tilde{A}_m Z \ , \tag{4.14}$$

where B is an $L^2 \times L^2$ matrix which subtracts the average of the elements in the vector $\tilde{A}_m Z$ from each element in $\tilde{A}_m Z$. That is, B is computed as

$$B = I_{L^2 \times L^2} - \frac{1}{L^2} O_{L^2 \times L^2} \ , \tag{4.15}$$

where $I_{L^2 \times L^2}$ is an $L^2 \times L^2$ identity matrix and $O_{L^2 \times L^2}$ is an $L^2 \times L^2$ matrix of ones.

To understand how we selected $\bar{X}$, note that $\tilde{A}_m Z$ is the part of $\hat{X}$ that depends upon Z. We think of $\bar{X}$ as a "centered" version of the part of $\hat{X}$ that depends upon Z, since B renders $\bar{X}$ invariant to the average of the elements in $\tilde{A}_m Z$. By splitting on $\bar{X}$, we are forcing the splits to depend on the differences among the elements in $\hat{X}$, rather than on the elements of $\hat{X}$ themselves. Geometrically, this leads to more classes for structures based on differences, such as edges of different orientation. In Sec. 4.4 we demonstrate that this can significantly improve interpolation quality.

The desired decision rule (4.10) is obtained as

$$e'_{\ell-1} = \left(\hat{e}^t_{\bar{X}|m} B \tilde{A}_m \right)^t \ , \tag{4.16}$$

and

$$\mu'_{\ell-1} = \hat{\mu}_{Z|m} \ , \tag{4.17}$$

where $\hat{e}_{\bar{X}|m}$ is the eigenvector corresponding to the largest eigenvalue of an estimate $\hat{\Sigma}_{\bar{X}|m}$ of the covariance matrix of $\bar{X}$,

$$\Sigma_{\bar{X}|m} = \mathrm{E}\left[\left(\bar{X} - \mu_{\bar{X}|m} \right) \left(\bar{X} - \mu_{\bar{X}|m} \right)^t | Z \in \mathcal{Z}_m \right] \ , \tag{4.18}$$

$$\mu_{\bar{X}|m} = \mathrm{E}\left[\bar{X} | Z \in \mathcal{Z}_m \right] \ , \tag{4.19}$$

and $\hat{\mu}_{Z|m}$ is an estimate of

$$\mu_{Z|m} = \mathrm{E}\left[Z|Z \in \mathcal{Z}_m\right] \ . \tag{4.20}$$

Observe that while this decision rule splits realizations of $\bar{X}$, it also readily yields a hyperplane which splits realizations of Z in $\mathcal{Z}$, since

$$\hat{e}^t_{\bar{X}|m}\left(\bar{X} - \hat{\mu}_{\bar{X}|m}\right) = \hat{e}^t_{\bar{X}|m}\left(B\tilde{A}_m Z - B\tilde{A}_m\hat{\mu}_{Z|m}\right) = \left(e'_{\ell-1}\right)^t\left(Z - \mu'_{\ell-1}\right) \ , \tag{4.21}$$

where $\hat{\mu}_{\bar{X}|m}$ is an estimate of $\mu_{\bar{X}|m}$. The idea behind the form of this split is to divide the cluster with a hyperplane passing through the mean of $\bar{X}$ and perpendicular to the direction in which $\bar{X}$ varies the most. Using second-order statistics the best way to estimate that direction is to take the eigenvector $e_{\bar{X}|m}$ which corresponds to the largest eigenvalue of $\Sigma_{\bar{X}|m}$ [25].

We obtain the desired vectors $\hat{e}_{\bar{X}|m}$ and $\hat{\mu}_{Z|m}$ as follows. First we compute $\hat{\mu}_{Z|m}$ as the sample mean of the z's in G_m:

$$\hat{\mu}_{Z|m} = \frac{1}{N_{G_m}} \sum_{i=0}^{N_{G_m}-1} z_i \ , \tag{4.22}$$

where N_{G_m} is the number of training vector pairs in G_m. We obtain $\hat{e}_{\bar{X}|m}$ by computing $\hat{\Sigma}_{\bar{X}|m}$ and then performing an eigenvalue decomposition. We obtain $\hat{\Sigma}_{\bar{X}|m}$ as

$$\hat{\Sigma}_{\bar{X}|m} = B\tilde{A}_m\hat{\Sigma}_{Z|m}\tilde{A}^t_m B^t \ , \tag{4.23}$$

where $\hat{\Sigma}_{Z|m}$ is an estimate of

$$\Sigma_{Z|m} = \mathrm{E}\left[(Z - \mu_{Z|m})(Z - \mu_{Z|m})^t | Z \in \mathcal{Z}_m\right] \ , \tag{4.24}$$

computed as

$$\hat{\Sigma}_{Z|m} = \frac{1}{N_{G_m}} \sum_{i=0}^{N_{G_m}-1} (z_i - \hat{\mu}_{Z|m})(z_i - \hat{\mu}_{Z|m})^t \ . \tag{4.25}$$

Generating interpolation filters for the candidate terminal nodes. With the candidate decision rule (4.10) computed, we separate the set G_m into two sets

$$G'_m = \left\{(x,z) \in G_m : \left(e'_{\ell-1}\right)^t\left(z - \mu'_{\ell-1}\right) < 0\right\} = \{(x_i, z_i)\}_{i=0}^{N_{G'_m}-1} \tag{4.26}$$

and

$$G'_\ell = \left\{(x,z) \in G_m : \left(e'_{\ell-1}\right)^t \left(z - \mu'_{\ell-1}\right) \geq 0\right\} = \{(x_i, z_i)\}_{i=0}^{N_{G'_\ell}-1} \ . \tag{4.27}$$

where $N_{G'_m}$ and $N_{G'_\ell}$ are the numbers of elements in G'_m and G'_ℓ, respectively. Note that we use the indices m and ℓ since they will be the indices of the new terminal nodes in $T(\ell+1)$. The next step is to compute candidate interpolation filters (4.11) and (4.12) for the candidate terminal nodes. Since the procedures are the same, we will only demonstrate for G'_m.

Our approach is to use maximum likelihood estimates of the parameters for the regression of X on $(Z|Z \in \mathcal{Z}'_m)$, where

$$\mathcal{Z}'_m = \left\{z \in \mathcal{Z}_m : \left(e'_{\ell-1}\right)^t \left(z - \mu'_{\ell-1}\right) < 0\right\} \ . \tag{4.28}$$

This is a well-known result from the multivariate statistics [33]. The desired interpolation filters are computed as

$$\tilde{A}'_m = \hat{\Sigma}'_{XZ|m} \left(\hat{\Sigma}'_{Z|m}\right)^{-1} \tag{4.29}$$

and

$$\tilde{\beta}'_m = \hat{\mu}'_{X|m} - \hat{\Sigma}'_{XZ|m} \left(\hat{\Sigma}'_{Z|m}\right)^{-1} \hat{\mu}'_{Z|m} \ , \tag{4.30}$$

where

$$\hat{\mu}'_{Z|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} z_i \ , \tag{4.31}$$

$$\hat{\mu}'_{X|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} x_i \ , \tag{4.32}$$

$$\hat{\Sigma}'_{Z|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} \left(z_i - \hat{\mu}'_{Z|m}\right)\left(z_i - \hat{\mu}'_{Z|m}\right)^t \ , \tag{4.33}$$

$$\hat{\Sigma}'_{XZ|m} = \frac{1}{N_{G'_m}} \sum_{i=0}^{N_{G'_m}-1} \left(x_i - \hat{\mu}'_{X|m}\right)\left(z_i - \hat{\mu}'_{Z|m}\right)^t \ , \tag{4.34}$$

and a superscript "-1" denotes taking the matrix inverse (or pseudo-inverse).

4.2.2.2 Pruning the tree-structured interpolator

The purpose of the pruning stage is to remove extraneous branches in $T(M_{\max})$, leaving behind a tree structure which efficiently represents a wide range of behaviors observable in images. Here we generate a decreasing sequence of pruned subtrees

$$T(M_{\max}), ..., T(M^*) \ , \tag{4.35}$$

$M_{\max} \geq M^* \geq 1$, where each tree is obtained from the preceding tree by removing a branch. We point out that that $M_{\max}$ must be specified by the person programming the algorithm, but M^* actually falls out of the algorithm by itself.

We employ a procedure which is similar to that used in [34] and in [35]. The basic idea is to start at the terminal nodes and to work our way up toward the root node, pruning any nonterminal node along the way which yields a decrease in the estimated mean-squared error $\mathcal{E}_P(\cdot)$. The only requirement on the order is that we never consider pruning at any nonterminal node until after we have considered pruning at both its children. Our approach is to submit the root node of $T(M_{\max})$ to a recursive subroutine that first considers pruning at each child of the current nonterminal node (with a subroutine call to itself), and then considers pruning at the current nonterminal node. One advantage of this procedure is that it yields the smallest optimally pruned subtree relative to the training set P [31].

After the pruning is complete, we regard the size and the shape of the final tree-structured interpolator T as fixed. That is,

$$M = M^* \ , \tag{4.36}$$

and the decision rules (4.5) for the nonterminal nodes in T are defined to be those in $T(M^*)$ (assuming that $M^* > 1$). The interpolation filters (4.6) are not defined to be those in $T(M^*)$, because we generate new ones after the pruning stage. We will describe this procedure in the next section.

4.2.2.3 Generating new interpolation filters

The last step in the training procedure is to compute the interpolation filters in T using a set D of interpolation training vector pairs which is very large relative to the sizes of the sets G and P. For example, in our experiments, the sets G and P each contained $300,000$ vector pairs, but D contained several million vector pairs. The idea here is to use many more vector pairs than were used to generate the interpolation filters for the classes originally, during the tree growing stage. This improves performance for interpolating behaviors which are distinct enough to have warranted the formation of classes, but which may have been slightly under-represented in the growing set. We have observed that this improves performance in particular for higher-order interpolations, such as for an interpolation factor $L = 4$.

This process is formally identical to the procedure by which we generate the candidate interpolation filters, which we explained in Sec. 4.2.2.1.

4.2.2.4 Obtaining the interpolation training sets

We obtain the interpolation training vector pairs from a set of training images which consists of low-resolution images and corresponding sharpened, high-resolution images. To generate the training images, we first procure a database of high-quality images from a photo CD image library. These are the input images in Fig. 4.5. We create the low-resolution training images from the input images by low-pass filtering followed by downsampling by a factor of L, where L is the desired interpolation factor. This is implemented using simple $L \times L$ block-averaging. Next, we create the high-resolution training images by sharpening the input images using unsharp masking with a sharpening factor of 1.0. By sharpening the high-resolution training images, we give the resulting interpolator a built-in sharpening effect.

Next we procure the interpolation training vector pairs which comprise the training sets G, P, and D. For any of these sets, we only need to select the pixels in the low-resolution training images at which we will procure the vector pairs, and then procure them using the process illustrated in Fig. 4.4. One requirement is that in

order for the pruning stage to be effective, the sets G and P should be extracted from different pixels in the training images. It is also important that the pixels in any given set be from spatially separate regions of the training images, so that the set represents a wide variety of image data.

We selected the sizes of sets G and P to be equal and as large as possible, under the constraint that during either of the growing and pruning stages, all of the training set should be maintained in computer memory along with the rest of the executable. This speeds up the training process by making it unnecessary for the computer to refer to the disk for a training vector pair each time it is needed. In our experiments, each of G and P contained $300,000$ interpolation training vector pairs. On the other hand, the training set D can be arbitrarily large, since each training vector pair in D only has to be accessed once. In our experiments, D contained between 5 and 20 million vector pairs.

4.3 Enhanced Tree-based Resolution Synthesis

Interpolated images often exhibit artifacts, regardless of the method used to do the interpolation. These artifacts occur particularly on edges, in the form of blurriness, ringing, and "jaggies." A common way to reduce blurriness is to sharpen the interpolated image; but this can exacerbate the other artifacts. For an alternative approach to enhanced image interpolation, we introduce a two-stage method called Enhanced Tree-based Resolution Synthesis (ETRS). In the first stage, we generate the TRS interpolation, and in the enhancement stage, we make corrective adjustments to the TRS interpolation to generate the ETRS interpolation. These corrective adjustments are computed using an enhancement predictor which is trained to reduce artifacts that occur in TRS interpolations. The enhancement predictor works by producing local estimates of the sign and the magnitude of the interpolation error for stronger adjustment around edges and in texture regions, giving the ETRS interpolation sharper appearance with fewer artifacts.

The idea behind ETRS is illustrated in Fig. 4.6. First observe Fig. 4.6(a), where we have rendered an edge at one resolution. In Fig. 4.6(b), we illustrate the TRS interpolation of the edge, generated from a lower-resolution rendering. Note that TRS estimates the edge with a smooth transition, since this is the best solution from a mean-squared error perspective. However, from a visual perspective, it is blurry. During the enhancement stage of ETRS, we would use the TRS interpolation to generate the image of Fig. 4.6(c) which is sharper and has more realistic impact, even if the position of the original edge has not been recovered perfectly. The enhancement process works by adding corrective adjustments to the TRS interpolation. For the present example, the image of corrective adjustments appears in Fig. 4.6(d). Note that we have added a constant offset to this image since adjustments can be either positive or negative. Here, black represents adjustments which subtract from the TRS interpolation; white represents adjustments which add to it; and the middle gray level (over most of the image) represents making no adjustment at all.

An overview of the ETRS algorithm appears in Fig. 4.7. The mechanism which drives the enhancement process is a regression tree which we use to characterize the interpolation error at each pixel location. We will refer to it as the enhancement predictor. Before ETRS interpolation can be executed, we generate the parameters for the enhancement predictor in a training process which uses sample images as well as the TRS interpolator parameters. In the remainder of this section, we describe the enhancement process in Sec. 4.3.1; and in Sec. 4.3.2, we describe the training procedure by which we generate parameters for the enhancement predictor.

Again, when we refer to images in this section, we mean monochrome images with pixel values from the interval $[0, 255]$. For color images, we perform the enhancement on the luminance component of the TRS interpolation. That is, we compute the ETRS interpolation by adding adjustments to the luminance component of the TRS interpolation, with the adjustments computed from the luminance components of the TRS and ETRS interpolations. We compute the luminance component of a color

image as

$$\text{luminance component } = 0.299r + 0.517g + 0.114b \ , \tag{4.37}$$

where "r," "g," and "b" represent the red, green, and blue planes, respectively. (Note that the luminance component is also an image with pixel values from the interval $[0, 255]$.) To add an adjustment to the luminance component of the TRS interpolation, we add the adjustment to each of the red, green, and blue planes.

4.3.1 Enhancement stage of ETRS

We generate each pixel in the enhanced interpolation by taking the corresponding pixel in the TRS interpolation and adding an adjustment to it:

$$x_{\text{etrs}} = x_{\text{trs}} + \bar{\delta} \ , \tag{4.38}$$

where $\bar{\delta}$ is the adjustment, and where x_{etrs} and x_{trs} are corresponding pixels in the ETRS and TRS interpolations. In order for the adjustments to improve an edge by imparting a sharper, more continuous appearance, we have found that they must be spatially consistent, having the same sign on one side of the edge, but instantaneously "flipping" across the edge (see Fig. 4.6(d)). For this reason, each adjustment is computed based on estimates of the sign and the magnitude of the interpolation error, which are obtained using the enhancement predictor. The enhancement process is illustrated in Fig. 4.8.

To describe this procedure more precisely, we first define a random variable Δ to represent the prediction error, *i.e.*, the difference between a pixel X in a sharpened, original rendering, and the corresponding pixel X_{trs} in the TRS interpolation:

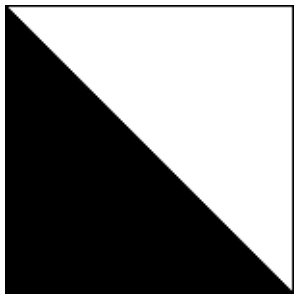
$$\Delta = X - X_{\text{trs}} \ . \tag{4.39}$$

We also define the random variables Ψ and Φ, to be respectively the sign of the interpolation error,

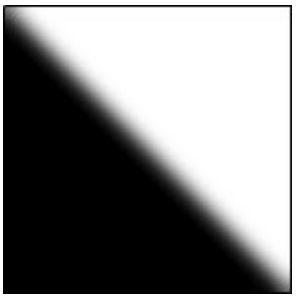
$$\Psi = \text{sign}\,(\Delta) \ , \tag{4.40}$$

and its magnitude,

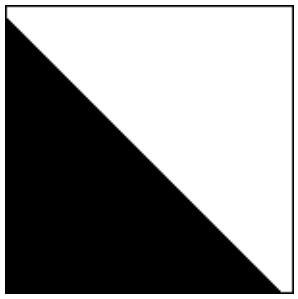
$$\Phi = |\Delta| \ . \tag{4.41}$$



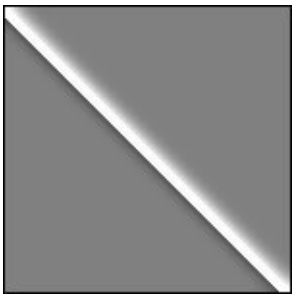
(a) Original edge.



(b) Illustration of the TRS interpolation.



(c) Illustration of the ETRS interpolation generated from the TRS interpolation of (b).



(d) Illustration of the adjustments added to the TRS interpolation during the enhancement stage.

Fig. 4.6. The idea behind ETRS.

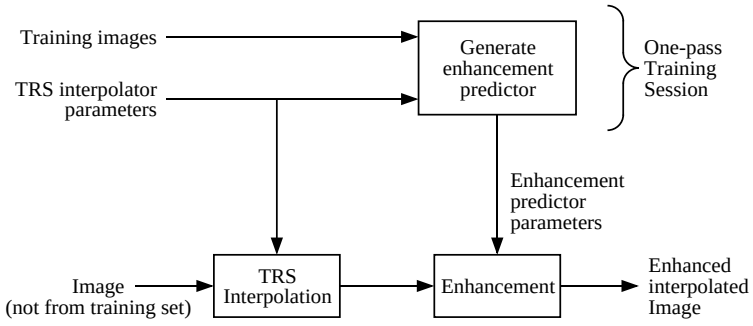


Fig. 4.7. Overview of the ETRS algorithm.

We estimate these three random variables using the enhancement predictor.

The input to this estimation is a vector which consists of pixels from a region centered at the current pixel in the TRS interpolation, along with previously generated pixels in the ETRS interpolation. We will model these pixels as elements of an H-dimensional random vector V. Figure 4.9 depicts the instantiation for V which we have used, where $H = 37$. (Note that since V contains previously generated ETRS pixels, we process the pixels in raster order.) We define the estimates generated using the enhancement predictor as the conditional expectations of Δ, Ψ, and Φ, given the realization of V extracted at the current pixel:

$$\hat{\delta} \stackrel{\text{def}}{=} \mathrm{E}\left[\Delta | V = v\right] \ , \tag{4.42}$$

$$\hat{\psi} \stackrel{\text{def}}{=} \mathrm{E}\left[\Psi | V = v\right] \ , \tag{4.43}$$

$$\hat{\phi} \stackrel{\text{def}}{=} \mathrm{E}\left[\Phi | V = v\right] \ , \tag{4.44}$$

where $\hat{\delta} \in R$; $\hat{\psi}$ is restricted to $[-1, 1]$; and $\hat{\phi}$ is restricted to being non-negative. We will describe how we actually compute $\hat{\delta}$, $\hat{\psi}$, and $\hat{\phi}$ below; but the process is formally identical to the TRS interpolation process.

With $\hat{\delta}$, $\hat{\psi}$, and $\hat{\phi}$ computed for the current pixel, we compute $\bar{\delta}$ as follows,

$$\bar{\delta} = \hat{\phi} f(\hat{\psi}) + \hat{\delta} \ , \tag{4.45}$$

where $f(\cdot)$ is defined as

$$f(\hat{\psi}) = \begin{cases} -1 & \text{if} \quad \hat{\psi} < -\tau \\ 0 & \text{if} \quad \left|\hat{\psi}\right| \leq \tau \\ 1 & \text{if} \quad \hat{\psi} > \tau \end{cases} \ , \tag{4.46}$$

with $\tau \geq 0$ being a fixed threshold parameter.

We arrived at the formula in (4.45) by looking at images of $\hat{\delta}, \hat{\psi}$, and $\hat{\phi}$ generated during the ETRS interpolation process. That is, we generated a separate monochrome image to display each of them. This allowed us to see $\hat{\delta}, \hat{\psi}$, and $\hat{\phi}$ within their own spatial contexts. We found that $\hat{\psi}$ tends to give good indications of the true sign of

the interpolation error; and moreover, it tends to take values with larger magnitudes on and around edges. We also found that $\hat{\phi}$ corresponds well with the magnitude of the interpolation error. These observations led us to the following strategy. At pixels where $\hat{\psi}$ has great enough magnitude to suggest that a strong adjustment is warranted, then we make adjustments which are more visible, and which generally have the sign of $\hat{\psi}$. At the other pixels, the adjustment is just $\hat{\delta}$, which tends to be small and imparts very little visual impact. The matter of whether $\hat{\psi}$ has great magnitude is mediated through the function $f(\cdot)$, which is a thresholding operation. This function is illustrated in Fig. 4.10; we have empirically determined a suitable value for τ to be 0.1.

The estimation by which we obtain $\hat{\delta}, \hat{\psi}$, and $\hat{\phi}$ is formally identical to the TRS interpolation process. Defining $\hat{y} = (\hat{\delta}\ \hat{\psi}\ \hat{\phi})^t$, we compute

$$\hat{y} = F_k v + \gamma_k \ . \tag{4.47}$$

Here, $k \in \{0, ..., N-1\}$ is the index of the class of v determined from U, with $1 \leq N < \infty$ being the number of classes in U; and F_k and γ_k are respectively the $3 \times H$ matrix and the 3-dimensional vector which comprise the prediction filter for class k. We compute k as $C_U(v)$, by taking the index of the terminal node at the end of a sequence of decisions of the form

$$g_n^t (v - \nu_n) \begin{array}{c} > \\ \\ < \end{array} 0 \ , \tag{4.48}$$

where g_n and ν_n are the H-dimensional vectors comprising the decision rule of the n-th nonterminal node in U. In practice, we have observed that rarely, but occasionally, the quantities computed in (4.47) can be invalid. This is not surprising since there is nothing which constrains $\hat{\psi}$ to equal the sign of $\hat{\delta}$, or $\hat{\phi}$ to equal $|\hat{\delta}|$. The two major violations are when $\hat{\psi} \notin [-1, 1]$, and when $\hat{\phi} < 0$. When this occurs we truncate the raw estimates to their closest valid values.

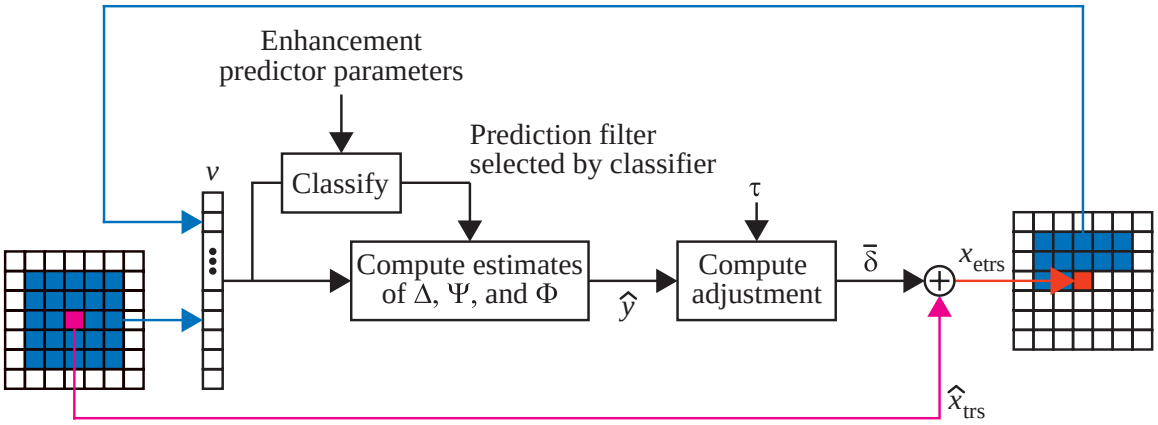


Fig. 4.8. Enhancement process used in ETRS.

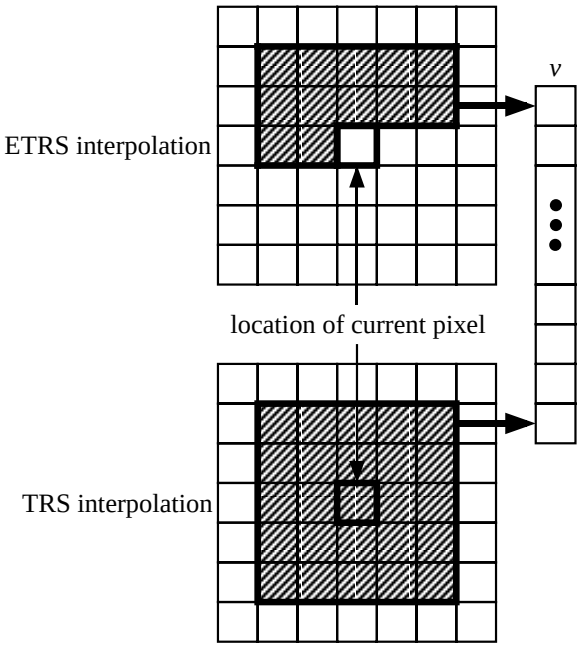


Fig. 4.9. The vector v used to characterize the interpolation error at the current pixel location in ETRS.

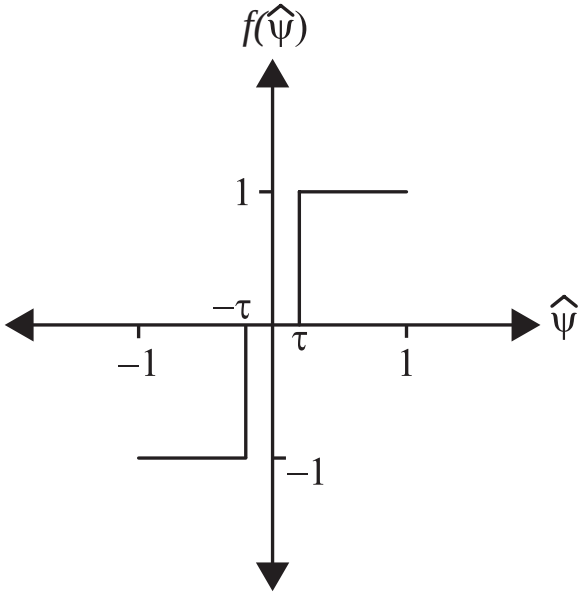


Fig. 4.10. Function $f(\cdot)$ used during the enhancement process to determine whether an adjustment should have great magnitude.

4.3.2 Generating parameters for the enhancement predictor: differences from the training for the TRS interpolator

Before the enhancement stage of ETRS can be executed, we generate the parameters for the regression tree U by training using sample images along with the TRS interpolator parameters. This process, which is illustrated in Fig. 4.11, is basically the same as that which we use to generate the TRS interpolator parameters; but there are three differences. The main difference deals with how we procure the training sets. This stems from the fact that in order to train the enhancement predictor, we need ETRS interpolations, which are not available until after an enhancement predictor has been obtained. The second difference is in how we split terminal nodes during the growing phase; and the third difference is in the resubstitution metric which we use to compare trees during the growing and pruning phases. We will describe these differences, rather than providing a repeat of Sec. 4.2.2; but we will maintain the notational framework of that section, so that subscripts, tildes, and primes have the same (or analogous) meanings. Explicitly, the parameters which we obtain from this process are the integer number $N \geq 1$ of classes represented in U; the decision rules

$$\{(g_n, \nu_n)\}_{n=0}^{N-2} \tag{4.49}$$

for the nonterminal nodes (assuming $N > 1$); and the prediction filters

$$\{(F_n, \gamma_n)\}_{n=0}^{N-1} \tag{4.50}$$

for the terminal nodes.

4.3.2.1 Method for obtaining the training sets

The training process for the enhancement predictor uses three sets of enhancement training vector pairs, which we denote as G, P, and D. We assume that each of these sets consists of independent realizations of (V, Y), where $Y = (\Delta\ \Psi\ \Phi)^t$. A generic enhancement training vector pair is illustrated in Fig. 4.12. Note that ideally, each enhancement training vector pair contains data from three versions of the same image: an original, sharpened rendering; a TRS interpolation; and an ETRS

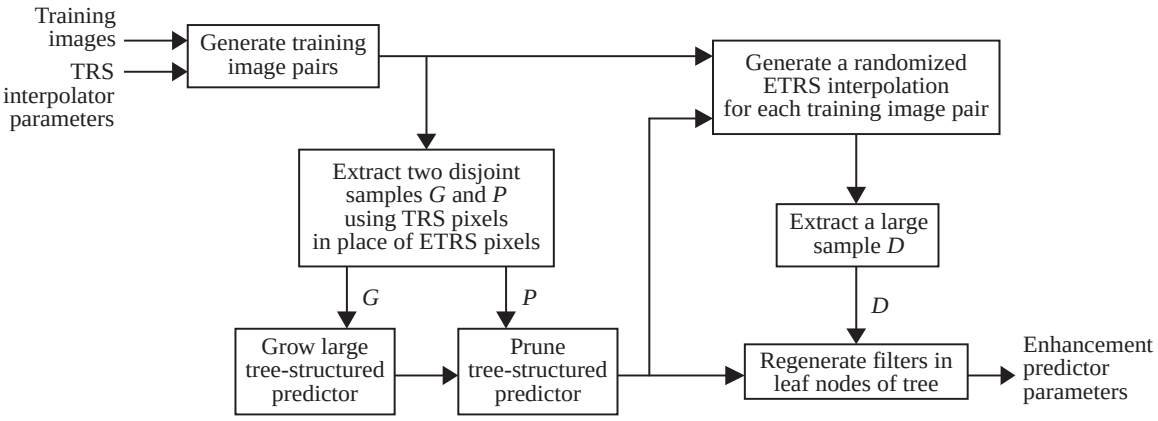


Fig. 4.11. Overview of the training process used to generate parameters for the ETRS enhancement predictor.

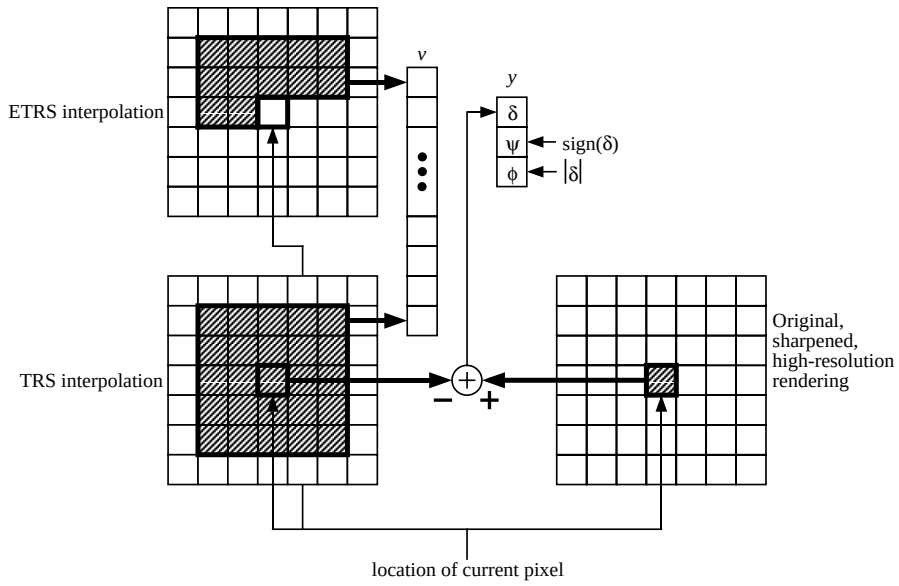


Fig. 4.12. A generic enhancement training vector pair (v, y).

interpolation. It is easy to generate the sharpened original rendering and the corresponding TRS interpolation. The catch is that during the training process, we do not have an enhancement predictor; so that it would be impossible to have true ETRS interpolations.

Our procedure is to use two types of images in place of the ETRS interpolations, depending upon which set of enhancement training vector pairs we are procuring.

1. For the first two sets, G and P, we use duplicates of the TRS interpolations in place of the ETRS interpolations. This means that each vector v in G and P contains some redundant pixels.

2. For the third set, D, we use randomized ETRS interpolations in place of the ETRS interpolations.

A randomized ETRS interpolation is different from a regular ETRS interpolation in that it is generated by adding random, rather than deterministic, adjustments to the pixels in the TRS interpolation. We will describe how we generate randomized ETRS interpolations below.

To create a pair consisting of one sharpened original image and one TRS interpolation of the same image, we use the process illustrated in Fig. 4.13. We sharpen the input training image to create the sharpened original image. Then to obtain the corresponding TRS interpolation, we generate a low-resolution rendering from the input training image by $L \times L$ block-averaging, and interpolate it back to the original resolution using TRS.

To generate a randomized ETRS interpolation, we follow a procedure which is very similar to the enhancement procedure described in Sec. 4.3.1. As stated above, the only difference is that the adjustments are random, rather than deterministic. Formally, denoting a random adjustment as δ_{rand}, we compute pixel $x_{\text{etrs,rand}}$ in the randomized ETRS interpolation as

$$x_{\text{etrs,rand}} = x_{\text{trs}} + \delta_{\text{rand}} \ , \tag{4.51}$$

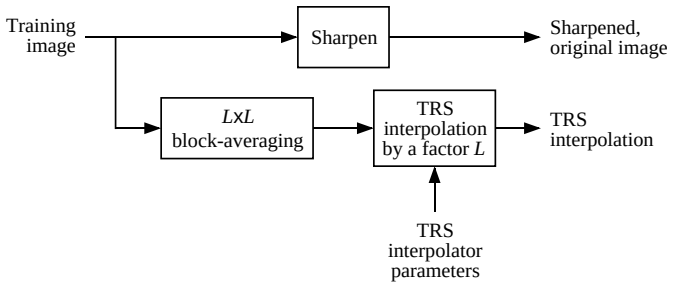


Fig. 4.13. Generation of a TRS interpolation and its original, sharpened counterpart.

where x_{trs} is the corresponding pixel in the TRS interpolation. We obtain δ_{rand} as a random sample drawn from the distribution of $(\Delta|V=v)$, the interpolation error conditioned on the vector v extracted at the current pixel.

We assume that the distribution of $(\Delta|V=v)$ has the probability density function (pdf) illustrated in Fig. 4.14. The form for this pdf results from the following assumption: given v, and provided that we know the sign of the interpolation error, the interpolation error is uniformly distributed in the direction of the sign. Writing the pdf of $(\Delta|V=v)$ as $p_{\Delta|V}(\cdot|v)$, we put this assumption more precisely as follows:

$$p_{\Delta|V}(\delta|v) = \begin{cases} -\frac{\pi_0}{2\alpha_0} & \text{if } 2\alpha_0 \le \delta < 0 \\ \frac{\pi_1}{2\alpha_1} & \text{if } 0 \le \delta \le 2\alpha_1 \\ 0 & \text{otherwise} \end{cases} , \tag{4.52}$$

where π_0, π_1, α_0, and α_1 are defined as follows:

$$\pi_0 = \Pr\{\Delta < 0 | V = v\} , \tag{4.53}$$

$$\pi_1 = \Pr\{\Delta \ge 0 | V = v\} , \tag{4.54}$$

$$\alpha_0 = \mathrm{E}[\Delta | \Delta < 0, V = v] , \tag{4.55}$$

$$\alpha_1 = \mathrm{E}[\Delta | \Delta \ge 0, V = v] . \tag{4.56}$$

Note here that by necessity, $\pi_0 \ge 0$, $\pi_1 \ge 0$, $\pi_0 + \pi_1 = 1$, $\alpha_0 \le 0$, and $\alpha_1 \ge 0$.

One way to obtain a random sample from this distribution would be to compute estimates of π_0, π_1, α_0, and α_1. However, it is equivalent to estimate $\mathrm{E}[\Psi|V=v]$, $\mathrm{E}[\Delta|V=v]$, and $\mathrm{E}[\Phi|V=v]$. Rather than using a "true" enhancement predictor to

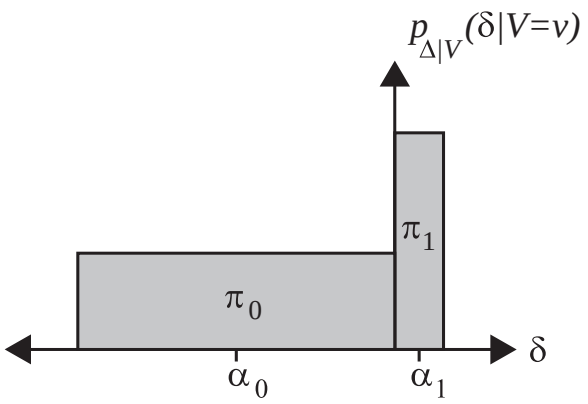


Fig. 4.14. Illustration of the form of the pdf for $(\Delta|V = v)$ which we assume for generating the randomized ETRS interpolations.

perform this estimation, we use the parameters which are the output of the pruning stage and the input to the filter-generation stage in Fig. 4.11. (They comprise a complete set of parameters for an enhancement predictor, even though they have not yet been through the filter generation procedure.) We will write these estimates with superscript tildes $(\tilde{\delta}, \tilde{\psi}, \tilde{\phi})$ to set them apart from estimates generated using a "true" enhancement predictor.

With $\tilde{\delta}, \tilde{\psi}$, and $\tilde{\phi}$ computed for the current pixel, we obtain δ_{rand} as

$$\delta_{\text{rand}} = \begin{cases} \left(\frac{2\left(1-\tilde{\psi}\right)-4u}{\left(1-\tilde{\psi}\right)^2}\right)\left(\tilde{\delta}-\tilde{\phi}\right) & \text{if } u \leq \frac{1}{2}\left(1-\tilde{\psi}\right) \\ \left(\frac{4u-2\left(1-\tilde{\psi}\right)}{\left(1+\tilde{\psi}\right)^2}\right)\left(\tilde{\delta}+\tilde{\phi}\right) & \text{else} \end{cases} , \tag{4.57}$$

where u is a random number drawn from a uniform $[0, 1]$ distribution. This may seem like an indirect procedure, considering that we already have an ETRS interpolation procedure which could be used here (*i.e.*, the deterministic procedure described in Sec. 4.3.1). However, the results which we have obtained using the procedure described here are a little better.

To see that $p_{\Delta|V}(\cdot|v)$ is completely characterized by $\mathrm{E}\left[\Psi|V=v\right], \mathrm{E}\left[\Delta|V=v\right]$, and $\mathrm{E}\left[\Phi|V=v\right]$, consider the following invertible system of equations. It consists of straightforward consequences of the form of $p_{\Delta|V}(\cdot|v)$:

$$\begin{bmatrix} -1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{pmatrix} \pi_0 \\ \pi_1 \\ \pi_0\alpha_0 \\ \pi_1\alpha_1 \end{pmatrix} = \begin{pmatrix} \mathrm{E}\left[\Psi|V=v\right] \\ 1 \\ \mathrm{E}\left[\Delta|V=v\right] \\ \mathrm{E}\left[\Phi|V=v\right] \end{pmatrix} . \tag{4.58}$$

By solving this system of equations and by substituting our estimates $\tilde{\delta}, \tilde{\psi}$, and $\tilde{\phi}$ for the expectations, we obtain the following estimates of π_0, π_1, α_0, and α_1, which we denote with superscript tildes:

$$\tilde{\pi}_0 = \frac{1}{2}(1-\tilde{\psi}) , \tag{4.59}$$

$$\tilde{\pi}_1 = \frac{1}{2}(1+\tilde{\psi}) , \tag{4.60}$$

$$\tilde{\alpha}_0 = \frac{1}{2\tilde{\pi}_0}(\tilde{\delta} - \tilde{\phi}) \ , \tag{4.61}$$

$$\tilde{\alpha}_1 = \frac{1}{2\tilde{\pi}_1}(\tilde{\delta} + \tilde{\phi}) \ . \tag{4.62}$$

Using these values in place of their respective parameters in (4.52), it is easy compute expression (4.51) which is used for the random sample generation.

4.3.2.2 Splitting rule

Another difference between the training procedures is in the vectors relative to which we split terminal nodes during the tree growing phases. In the case of training for the enhancement predictor, our objective is to establish classes for different types of edge regions, since the enhancement predictor is intended to predict error in those regions.

To achieve this objective, we split relative to a vector $\bar{V}$, defined as

$$\bar{V} = BV \ . \tag{4.63}$$

Here, B is an $H \times H$ matrix which subtracts the average of the elements in V from each of the elements in V, computed as

$$B = I_{H\times H} - \frac{1}{H} O_{H\times H} \ , \tag{4.64}$$

where $I_{H\times H}$ is an $H \times H$ identity matrix and $O_{H\times H}$ is an $H \times H$ matrix of ones. Note that this is similar to the vector $\bar{X}$ relative to which we determine the splits for the TRS interpolator: the main idea is to make splits based on the differences among the elements in V rather than on the elements of V themselves. This makes the splitting process more efficient, since successive splits generate classes for different edge behaviors rather than being wasted on classes for behaviors which are structurally more similar.

To obtain the splitting rule we compute

$$g'_{\ell-1} = \left(\hat{e}^t_{\bar{V}|n} B\right)^t \ , \tag{4.65}$$

and

$$\nu'_{\ell-1} = \hat{\mu}_{V|n} \ , \tag{4.66}$$

where $\hat{e}_{\bar{V}|n}$ is the eigenvector corresponding to the largest eigenvalue of an estimate $\hat{\Sigma}_{\bar{V}|n}$ of the covariance matrix $\Sigma_{\bar{V}|n}$. We first compute $\hat{\mu}_{V|n}$ as follows:

$$\hat{\mu}_{V|n} = \frac{1}{N_{G_n}} \sum_{i=0}^{N_{G_n}-1} v_i \ . \tag{4.67}$$

Next we obtain $\hat{e}_{\bar{V}|n}$ by computing $\hat{\Sigma}_{\bar{V}|n}$ and then performing an eigenvalue decomposition. We obtain $\hat{\Sigma}_{\bar{V}|n}$ as

$$\hat{\Sigma}_{\bar{V}|n} = B\hat{\Sigma}_{V|n}B^t \ , \tag{4.68}$$

where $\hat{\Sigma}_{V|n}$ is computed as

$$\hat{\Sigma}_{V|n} = \frac{1}{N_{G_n}} \sum_{i=0}^{N_{G_n}-1} (v_i - \hat{\mu}_{V|n})(v_i - \hat{\mu}_{V|n})^t \ . \tag{4.69}$$

4.3.2.3 Resubstitution metric

The third difference between the training procedures for the enhancement and interpolation regression trees lies in the resubstitution metrics which we use to evaluate them during either of the growing or pruning phases. For the case of the enhancement predictor, we compute the sample mean-squared prediction error for Δ, and we use the notation $\mathcal{F}_S(U(\ell))$, where S can equal either G or P. We evaluate $\mathcal{F}_S(U(\ell))$ by adding up the contributions of each of the terminal nodes, with the contribution of the n-th terminal node computed as

$$\mathcal{F}_{S_n}(U(\ell)) = \frac{1}{N_S} \sum_{i=0}^{N_{S_n}-1} \left(\delta_i - \tilde{\delta}_i\right)^2 \ , \tag{4.70}$$

where δ_i is the first element in y_i, and $\tilde{\delta}_i$ is the first element in $(\tilde{F}_n v_i + \tilde{\gamma}_n)$.

4.4 Results

In this section we present results of interpolation by a factor of $L = 2$, rendered at 75 dots per inch. Note that none of the images which appear in this section was among

the training images which we used to generate the parameters for the interpolator or for the enhancement predictor.

In Fig. 4.15, we compare TRS to pixel replication and bilinear interpolation. Note that the TRS interpolation is sharper than the bilinear result, and the edges in the TRS result are more continuous than they are in the pixel replication result. We have also included Fig. 4.16, to follow up on the part of Sec. 4.2.2.1 where we describe our method for splitting terminal nodes. Here we demonstrate the effect of centering the vector relative to which we determine the splits, using the matrix B. Note that the result generated by splitting on a vector which was not centered exhibits objectionable "jaggie" artifacts around edges. These artifacts are not nearly so prominent in the standard TRS result. An interesting fact is that although the same training sets were used to generate the two sets of interpolator parameters, the interpolator parameters for standard TRS had only 23 classes, while the parameters generated without the centering had 40 classes. This implies that the centering makes the splitting process more efficient by encouraging the formation of visually distinct classes such as edges of different orientation, rather than wasting splits by defining classes for behaviors which are visually more similar.

We have also compared the results of TRS with those of Resolution Synthesis (RS) from Chap. 2. (Actually the RS results shown here were generated using Efficient RS as it is described in Sec. 2.2.) In this comparison, the numbers of classes in the sets of interpolator parameters (for TRS and for RS) play an important role. For relatively low numbers of classes (say, around 25), the two methods tend to perform about equally; but as the numbers of classes increase, RS interpolation quality tends to improve more significantly and more quickly. This is illustrated in Figs. 4.17 and 4.18. In Fig. 4.17, the RS and TRS results are of basically the same interpolation quality; and both the parameter sets used for this figure contain the same number of classes (23). But in Fig. 4.18, the RS result exhibits fewer interpolation artifacts than does the TRS result, although the RS parameter set used for this figure contains far fewer classes (96 for RS, and 500 for TRS). (To generate a TRS interpolator with

500 classes, we used the training algorithm described in Sec. 4.2 with the pruning part of the algorithm turned off.)

One factor which contributes to the difference between TRS and RS is that there is no mechanism in TRS which is completely analogous to the projection operator for cluster vectors used in RS. (Recall that using the projection operator in RS led to improved interpolation quality by encouraging the formation of more classes for visually distinct behaviors such as edges.) Developing an analogous mechanism within the framework of TRS could be a subject for future research, and it could possibly change the overall comparison. This could lead to significant benefits in terms of computational cost, for reasons involving both the filter selection and the filtering. In RS, doubling the number of classes also doubles the computation required to select which filter(s) to use for the interpolation; while in TRS, it only adds an additional vector comparison to the filter selection process, assuming an approximately balanced tree structure. Also, RS generally uses more than one filter per input pixel, while TRS uses exactly one filter per input pixel.

Continuing with the comparison of TRS and RS, it is also interesting that the RS interpolator parameter sets used for Figs. 4.17 and 4.18 were generated using much less training data. In fact, to generate the parameters for the RS classifiers (the Gaussian mixture models), we used $60,000$ cluster vectors; while to generate the parameters for the TRS classifiers (the nonterminal nodes in the binary trees), we used $600,000$ and $300,000$ training vector pairs, respectively for TRS with 23 (Fig. 4.17) and 500 (Fig. 4.18) classes.

We demonstrate ETRS interpolation in Figs. 4.19 and 4.20. Figure 4.19 shows a TRS interpolation and the ETRS interpolation which was generated from it; while Fig. 4.20 shows the image of adjustments which were added to the TRS interpolation during the enhancement process. We have added a constant offset to the image of Fig. 4.20, since the adjustments can be either positive or negative. We observe that the ETRS interpolation is sharper and exhibits fewer objectionable artifacts than the TRS interpolation, due to the adjustments which were made on and around edges.

For these results, we have used $\tau = 0.1$. By modifying the value of τ we would change the set of pixels where adjustments are imparted. If we increase τ, then this set shrinks, resulting in adjustment only on and around the strongest edges in the interpolated image.

Finally, in Fig. 4.21, we include a comparison of all three interpolation algorithms introduced in this thesis: RS, TRS, and ETRS. This figure reflects our observation that overall, both RS and ETRS tend to perform better than TRS. The comparison between RS and ETRS is more interesting. One observation is that many of the edges in the ETRS result are suitably continuous (*i.e.*, free of "jaggie" artifacts), and are considerably sharper than the same edges in the RS result. However, it could also be said that the RS result is more spatially consistent in terms of sharpness. This leads to a more natural appearance, which is why the RS result is the best among the three in this figure, in the opinion of the author.

4.5 Conclusions

In this report, we have introduced TRS and ETRS, two methods for image interpolation. One advantage of TRS is that it has a simple implementation. But we have also seen that since the TRS interpolator is designed to approximate the conditional mean estimator of the high-resolution pixels given corresponding low-resolution pixels, it provides an approximately optimal (MMSE) interpolation. We have also introduced ETRS which includes a second enhancement stage. The purpose of this enhancement stage is to correct for artifacts in the TRS interpolation, and to impart additional enhancements. We have demonstrated the effectiveness of both TRS and ETRS by showing results of interpolating images which were not among the training images used to generate the predictor parameters.

(a) Pixel replication.

(b) Bilinear interpolation.

(c) TRS.

Fig. 4.15. Results of interpolation by a factor of $L = 2$.

(a) Result of not centering the vector used to determine the splits.

(b) Standard TRS.

Fig. 4.16. The effect of centering the vector used to determine the decision rules during the interpolator design; interpolation by a factor of $L = 2$.

(a) TRS using 23 classes.

(b) RS using 23 classes.

Fig. 4.17. Comparison of interpolation by a factor of $L = 2$, using (a) TRS and (b) RS. Both sets of interpolator parameters used here contain 23 classes.

(a) TRS using 500 classes.

(b) RS using 96 classes.

Fig. 4.18. Comparison of interpolation by a factor of $L = 2$, using (a) TRS and (b) RS. The TRS interpolator parameters used for (a) contain 500 classes, while the RS interpolator parameters used for (b) contain 96 classes.

(a) TRS. (b) ETRS.

Fig. 4.19. TRS and ETRS interpolations by a factor of $L = 2$.

Fig. 4.20. Image of adjustments added to the TRS interpolation to generate the ETRS interpolation in Fig. 4.19.

(a) RS.

(b) TRS.

(c) ETRS.

Fig. 4.21. Comparison of interpolations by a factor of $L = 2$, generated by (a) RS, (b) Tree-based RS, and (c) Enhanced Tree-based RS.

LIST OF REFERENCES

[1] H. S. Hou and H. C. Andrews, "Cubic splines for image interpolation and digital filtering," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 6, pp. 508–517, December 1978.

[2] M. Unser, A. Aldroubi, and M. Eden, "Fast B-spline transforms for continuous image representation and interpolation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 3, pp. 277–285, March 1991.

[3] M. Unser, A. Aldroubi, and M. Eden, "Enlargement or reduction of digital images with minimum loss of information," *IEEE Transactions on Image Processing*, vol. 4, no. 3, pp. 247–258, March 1995.

[4] R. G. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, December 1981.

[5] A. K. Jain, *Fundamentals of Digital Image Processing*, pp. 253–255. Englewood Cliffs, New Jersey: Prentice Hall, 1989.

[6] V. R. Algazi, G. E. Ford, and R. Potharlanka, "Directional interpolation of images based on visual properties and rank order filtering," *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, 1991, pp. 3005 – 3008.

[7] G. E. Ford, R. R. Estes, and H. Chen, "Space scale analysis for image sampling and digital interpolation," *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, 1992, pp. III–165 – III–168.

[8] B. Ayazifar and J. S. Lim, "Pel-adaptive model-based interpolation of spatially subsampled images," *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, 1992, pp. III–181 – III–184.

[9] K. Jensen and D. Anastassiou, "Subpixel edge localization and the interpolation of still images," *IEEE Transactions on Image Processing*, vol. 4, no. 3, pp. 285–295, March 1995.

[10] J. P. Allebach and P. W. Wong, "Edge-directed interpolation," *Proc. of the International Conference on Image Processing*, 1996, pp. 707 – 710.

[11] R. R. Schultz and R. L. Stevenson, "A Bayesian approach to image expansion for improved definition," *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 233–242, May 1994.

[12] K. Popat and R. W. Picard, "Cluster-based probability model and its application to image and texture processing," *IEEE Transactions on Image Processing*, vol. 6, no. 2, pp. 268–284, February 1997.

[13] L. L. Scharf, *Statistical Signal Processing*. Reading, MA: Addison-Wesley, 1991.

[14] C. R. Rao, *Linear Statistical Inference and Its Applications, Second Edition*, ch. 5. New York: John Wiley and Sons, 1973.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Stat. Society B*, vol. 39, pp. 1–38, 1977.

[16] C. F. J. Wu, "On the convergence properties of the EM algorithm," *The Annals of Statistics*, vol. 11, no. 1, pp. 95–103, 1983.

[17] R. A. Boyles, "On the convergence of the EM algorithm," *J. Roy. Stat. Society B*, vol. 45, no. 1, pp. 47–50, 1983.

[18] R. A. Redner and H. F. Walker, "Mixture densities, maximum likelihood, and the EM algorithm," *SIAM Review*, vol. 26, no. 2, pp. 195–239, April 1984.

[19] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, December 1974.

[20] J. Rissanen, "A universal prior for integers and estimation by minimum description length," *The Annals of Statistics*, vol. 11, no. 2, pp. 416–431, 1983.

[21] R. L. Kashyap, "Inconsistency of the AIC rule for estimating the order of autoregressive models," *IEEE Transactions on Automatic Control*, vol. 25, no. 5, pp. 996–998, October 1980.

[22] C. A. Bouman and M. Shapiro, "SMAP image segmentation software package user's manual." Appendix, April 1996.

[23] P. Heckbert, "Color image quantization for frame buffer display," *Computer Graphics*, vol. 16, no. 3, pp. 297–307, July 1982.

[24] S. J. Wan, S. K. M. Wong, and P. Prusinkiewicz, "An algorithm for multidimensional data clustering," *ACM Transactions on Mathematical Software*, vol. 14, no. 2, pp. 153–162, June 1988.

[25] M. T. Orchard and C. A. Bouman, "Color quantization of images," *IEEE Transactions on Signal Processing*, vol. 39, no. 12, pp. 2677–2690, December 1991.

[26] A. Gersho, "Optimal nonlinear interpolative vector quantization," *IEEE Transactions on Communications*, vol. 38, no. 9, pp. 1285–1287, September 1990.

[27] H.-M. Hang and B. G. Haskell, "Interpolative vector quantization of color images," *IEEE Transactions on Communications*, vol. 36, no. 4, pp. 465–470, April 1988.

[28] Y.-S. Ho and A. Gersho, "A variable-rate image coding scheme with vector quantization and clustering interpolation," *Proc. IEEE Global Commun. Conf.*, 1989, pp. 898–902.

[29] Y.-S. Ho and A. Gersho, "Variable-rate contour-based interpolative vector quantization for image coding," *Proc. IEEE Global Commun. Conf.*, 1988, pp. 1890–1893.

[30] S. G. Chang, Z. Cvetković, and M. Vetterli, "Resolution enhancement of images using wavelet transform extrema extrapolation," *Proc. of the International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, 1995, pp. 2379–2382.

[31] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, pp. 163–174, February 1991.

[32] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth International Group, 1984.

[33] T. W. Anderson, *An Introduction to Multivariate Statistical Analysis*. New York: John Wiley and Sons, 1958.

[34] S. B. Gelfand, C. S. Ravishankar, and E. J. Delp, "Tree-structured piecewise linear adaptive equalization," *IEEE Transactions on Communications*, vol. 41, no. 1, pp. 70–82, January 1993.

[35] S. B. Gelfand and C. S. Ravishankar, "A tree-structured piecewise linear adaptive filter," *IEEE Transactions on Information Theory*, vol. 39, no. 6, pp. 1907–1922, November 1993.

APPENDIX

In this appendix, we derive the filter design procedure, which amounts to the maximization of (2.28), with the function Q_1 defined in (2.23). To begin the derivation, we recall assumption 2, so that we can write Q_1 explicitly:

$$Q_1(\psi \mid \hat{\theta}) = \sum_{j=1}^{M} \sum_{s \in S} \log p_{x|z,j}(x_s \mid z_s, j, \psi) p_{j|y}(j \mid y_s, \hat{\theta}) \tag{A.1}$$

$$= \sum_{j=1}^{M} \sum_{s \in S} \log p_{x|z,j}(x_s \mid z_s, A_j, \beta_j, \Lambda_j) p_{j|y}(j \mid y_s, \hat{\theta}) \ . \tag{A.2}$$

We assume that the filter parameters $(A_J, \beta_J, \Lambda_J)$ for the different classes do not depend on each other, so that our task is made up of M separate maximizations. The j–th of these maximizations is

$$\hat{A}_j, \hat{\beta}_j, \hat{\Lambda}_j = \arg \max_{(A,\beta,\Lambda)} Q_{1,j}(A, \beta, \Lambda \mid \hat{\theta}) \ , \tag{A.3}$$

where $Q_{1,j}$ is defined as

$$Q_{1,j}(A, \beta, \Lambda \mid \hat{\theta})$$

$$= \sum_{s \in S} \log p_{x|z,j}(x_s \mid z_s, A, \beta, \Lambda) p_{j|y}(j \mid y_s, \hat{\theta}) \tag{A.4}$$

$$= \sum_{s \in S} \log \left(\frac{\exp\left(-\frac{1}{2} (x_s - Az_s - \beta)^t \Lambda^{-1} (x_s - Az_s - \beta)\right)}{(2\pi)^{\frac{L^2}{2}} |\Lambda|^{\frac{1}{2}}} \right) p_{j|y}(j \mid y_s, \hat{\theta}) \tag{A.5}$$

$$= -\frac{L^2 N_j}{2} \log 2\pi + \frac{N_j}{2} \log \left|\Lambda^{-1}\right| - \frac{1}{2} \xi_j(A, \beta, \Lambda \mid \hat{\theta}) \ ; \tag{A.6}$$

here, L is the interpolation factor, N_j is defined in (2.37), and $\xi_j(A, \beta, \Lambda \mid \hat{\theta})$ is defined to be

$$\xi_j(A, \beta, \Lambda \mid \hat{\theta}) = \sum_{s \in S} (x_s - Az_s - \beta)^t \Lambda^{-1} (x_s - Az_s - \beta) p_{j|y}(j \mid y_s, \hat{\theta}) \ . \tag{A.7}$$

We compute $\hat{A}_j, \hat{\beta}_j$, and $\hat{\Lambda}_j$ as the values of these parameters at which the derivatives of $Q_{1,j}$ vanish. While this is only a necessary condition for (A.3), it may be verified that the computed parameters are in fact maximizers of $Q_{1,j}$. The key to computing the required derivatives of $Q_{1,j}$ lies in writing ξ_j in various convenient forms, which may be obtained through some straightforward manipulations.

The optimal estimates of A and β turn out to be functions of one another. To find their simultaneous solution, we first compute the estimate of β as a function of A. For this, we write ξ_j as

$$\begin{aligned}
&\xi_j(A, \beta, \Lambda \mid \hat{\theta}) \\
&\quad = \quad \text{trace} \left(\Lambda^{-1} \sum_{s \in S} (x_s - A z_s - \beta)(x_s - A z_s - \beta)^t p_{j|y}(j \mid y_s, \hat{\theta}) \right) \qquad \text{(A.8)} \\
&\quad = \quad N_j \, \text{trace} \left(\Lambda^{-1} \left(\Gamma_{xx|j} - \Gamma_{xz|j} A^t - A \Gamma_{xz|j}^t + A \Gamma_{zz|j} A^t \right) \right) \qquad \text{(A.9)} \\
&\qquad + N_j (\nu_{x|j} - A \nu_{z|j} - \beta)^t \Lambda^{-1} (\nu_{x|j} - A \nu_{z|j} - \beta) \ ,
\end{aligned}$$

where $\Gamma_{xx|j}, \Gamma_{xz|j}, \Gamma_{zz|j}, \nu_{x|j}$, and $\nu_{z|j}$ are defined in (2.40) and (2.38). From this, we compute the row vector $\nabla_\beta Q_{1,j}$ of derivatives, whose m–th element is the derivative of $Q_{1,j}$ with respect to the m–th element in β:

$$\begin{aligned}
\nabla_\beta Q_{1,j}(A, \beta, \Lambda \mid \hat{\theta}) \quad &= \quad -\frac{1}{2} \nabla_\beta \xi_j(A, \beta, \Lambda \mid \hat{\theta}) \qquad \text{(A.10)} \\
&= \quad N_j (\nu_{x|j} - A \nu_{z|j} - \beta)^t \Lambda^{-1} \ . \qquad \text{(A.11)}
\end{aligned}$$

Setting this equal to zero and solving yields our estimate of β in terms of A:

$$\hat{\beta}_j(A) = \nu_{x|j} - A \nu_{z|j} \ . \qquad \text{(A.12)}$$

We use this functional form for β to find our estimate of A. Inserting (A.12) into (A.9), we again write ξ_j,

$$\begin{aligned}
&\xi_j(A, \hat{\beta}_j(A), \Lambda \mid \hat{\theta}) \\
&\quad = \quad N_j \, \text{trace} \left(\Lambda^{-1} \left(\Gamma_{xx|j} - \Gamma_{xz|j} A^t - A \Gamma_{xz|j}^t + A \Gamma_{zz|j} A^t \right) \right) \ . \qquad \text{(A.13)}
\end{aligned}$$

From here, it is straightforward to compute the matrix $\nabla_A Q_{1,j}$ of derivatives, whose m, n–th element is the derivative of $Q_{1,j}$ with respect to the m, n–th element of A:

$$\nabla_A Q_{1,j}(A, \hat{\beta}_j(A), \Lambda \mid \hat{\theta}) \quad = \quad -\frac{1}{2} \nabla_A \xi_j(A, \hat{\beta}_j(A), \Lambda \mid \hat{\theta}) \qquad \text{(A.14)}$$

$$= \quad -N_j \Lambda^{-1} \left(A\Gamma_{zz|j} - \Gamma_{xz|j} \right) \ . \tag{A.15}$$

Setting this equal to zero and solving for A yields (2.42). Plugging the estimate for A back into (A.12) yields (2.43).

It remains to compute $\hat{\Lambda}_j$. We first simplify ξ_j by inserting our estimates for A and β into (A.13):

$$\xi_j(\hat{A}_j, \hat{\beta}_j, \Lambda \mid \hat{\theta}) \quad = \quad N_j \, \text{trace} \left(\Lambda^{-1} \left(\Gamma_{xx|j} - \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \Gamma_{xz|j}^t \right) \right) \ , \tag{A.16}$$

so that $Q_{1,j}$ may be written as

$$\begin{aligned} & Q_{1,j}(\hat{A}_j, \hat{\beta}_j, \Lambda \mid \hat{\theta}) \\ & \quad = \quad -\frac{L^2 N_j}{2} \log 2\pi + \frac{N_j}{2} \log \left| \Lambda^{-1} \right| - \frac{N_j}{2} \sum_{m=1}^{L^2} \sum_{n=1}^{L^2} \lambda^{m,n} g_{m,n} \ , \end{aligned} \tag{A.17}$$

where we have used $g_{m,n}$ to denote the m,n–th element of the matrix

$$\Gamma_{xx|j} - \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \Gamma_{xz|j}^t .$$

From this representation, we can compute the matrix $\nabla_{\Lambda^{-1}} Q_{1,j}$ of derivatives, whose m,n–th element is the derivative of $Q_{1,j}$ with respect to the m,n–th element of Λ^{-1}. The only obstacle here is the derivative of the determinant with respect to the m,n–th element of Λ^{-1}, which may be computed by writing the cofactor expansion for the determinant, along the m–th row of Λ^{-1}. The resulting derivative is

$$\nabla_{\Lambda^{-1}} Q_{1,j}(\hat{A}_j, \hat{\beta}_j, \Lambda \mid \hat{\theta}) \quad = \quad \frac{N_j}{2} \Lambda - \frac{N_j}{2} \left(\Gamma_{xx|j} - \Gamma_{xz|j} \Gamma_{zz|j}^{-1} \Gamma_{xz|j}^t \right) \ . \tag{A.18}$$

Setting this equal to zero and solving for Λ implies (2.44).

VITA

Clayton Brian Atkins was born in Frankfurt, (West) Germany, on January 29, 1969. When he was two, he moved to Indianapolis, IN, where he lived until he started college in 1987. He was graduated from Purdue University at West Lafayette, IN, with the degrees of B.S.E.E. (1992); M.S.E.E. (1994), and Ph.D. (1998). During his undergraduate curriculum, he participated in the cooperative education program by working for Rockwell International Corp., in Cedar Rapids, IA. He now lives in Mountain View, CA, and works at Hewlett-Packard Laboratories, in Palo Alto, CA.