Compressing the Illumination-Adjustable Images with Principal Component Analysis

Pun-Mo Ho

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

December, 2002

Abstract

The ability to change illumination is a crucial factor in image-based modeling and rendering. Image-based relighting offers such capability. However, the trade-off is the enormous increase of storage requirement. In this thesis, we propose a compression scheme that effectively reduces the data volume while maintaining the real-time relighting capability. The proposed method is based on principal component analysis (PCA). A block-wise PCA is used to practically process the huge input data. The output of PCA is a set of eigenimages and the corresponding relighting coefficients. By dropping those low-energy eigenimages, the data size is drastically reduced. To further compress the data, eigenimages left are compressed using transform coding and quantization while the relighting coefficients are compressed using uniform quantization. We also suggest the suitable target bit rate for each phase of the compression method in order to preserve the visual quality. Finally, we proposed real-time engine that relights images from the compressed data.

摘要

在基於圖像的模型與繪制技術中，能夠控制光照是很重要的。而基於圖像的光照技術就能提供此項功能。然而，儲存所需就會因此大大增加。在這論文中，我們提議一個能夠有效地減低儲存需要而同時能夠維持實時光照能力的壓縮方案。我們提議的這個方案是基於主要成份分析。在實踐中，首先我們會用基於塊的主要成份分析來初步減低資料的分量。經此步驟後，我們會得到一系列的本徵圖像與其光照系數。若我們把低能量的本徵圖像與其光照系數扔掉，那末原來的資料的分量就能夠大大地減少。為了能進一步壓縮資料，保留的本徵圖像會進行交換編碼與量化，而保留的光照系數就會被均勻量化。在這論文中，我們亦為每個壓縮步驟提供一個可維持視覺質素的位速率。最後，我們提議一個能夠用由以上方案壓縮而成的資料來重新光照圖像的實時光照器。

Acknowledgments

First, I wish to express my gratitude to my supervisor Prof. Tien-Tsin Wong for giving continuous support and guidance in my research. I would like to thank Dr. Chi-Sing Leung, Dr. Siu-Hang Or, Prof. Chi-Shing Lui, and Prof. Ho-Man Lee for giving me helpful advice. I would also like to thank my colleagues, Tak-Fu Tung, Tsz-Yeung Wong, Cheuk-Man Lee, Wai-Man Pang, and others who have helped me in these two years. I would like to express my deepest gratitude to my parents and family. Finally, I would like to give special thanks to my girl friend, Maggie, and my lovely hamsters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Image-based relighting [1, 2, 3] offers the image-based entities, such as light field [4], lumigraph [4, 5], concentric mosaic [6] and panorama [7], an ability to change the illumination. Image-based relighting can be used in computer games. It always has chance that the background of a computer game has to be changed constantly, for example from day to night. As the quality of modern computer games is demanding, loads of beautiful scenes are required. This hinders the smoothness of the computer game if the game has to stop every time a scene needed to be rendered. Image-based relighting shows its usefulness because rendering time for image-based relighting is independent of the scene complexity. Otherwise, thousands of triangles have to be rendered if we use traditional graphics approach. Image-based relighting depends on samples of a scenery illuminated under different lighting directions. Given such set of images, we can generate the scenery under any desired directions. Figure 1.1 illustrates this idea. Through image-based relighting, we can generate images with light sources locating at the white dots, where samples are taken, as well as the red dots, where samples are not taken originally.

Since the representation is image-based and no geometry is used in the

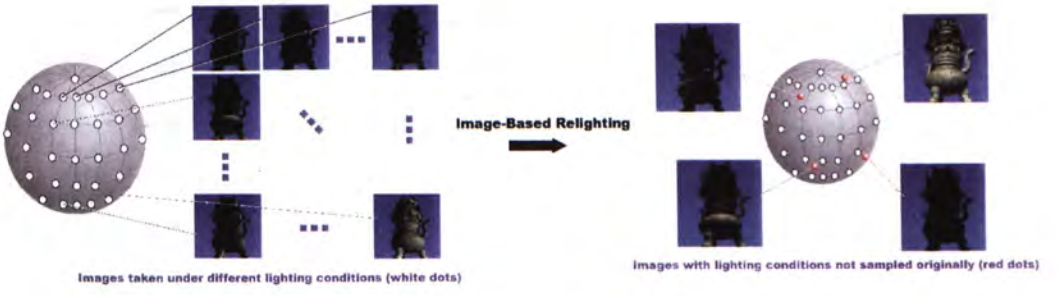


Figure 1.1: Image-based relighting.

computation of reflected radiance, the rendering (*relighting*) time is independent of scene complexity and can be in real-time. However, the data size is enormous when the illumination information is included in the image-based representation. For a 1024×256 color image captured under 1,200 illumination conditions, 900MB storage size is required if no compression is done. Therefore, data compression is a must.

1.2 Existing Approaches

There have been several work [4, 8, 9] in compressing data for image-based navigation such as light field and panorama. It seems that compression methods for image-based navigation may also be applied for image-based relighting. However, there is a fundamental difference between the nature of image data from these two categories. Data for image-based navigation is more related to geometry while data for image-based relighting is more related to surface reflectance.

The simplest way to compress a collection of images is to use still image coding such as JPEG and JPEG 2000. They are proven to be the most effective way for compressing still images by their widely usage in still image coding nowadays. However, they are not effective in compressing illumination-adjustable images because they cannot make use of the data correlation among

any two individual images. In other words, they cannot make use of the data correlation along the lighting domain. In order to make use the data correlation along the lighting domain, we can use MPEG to compress illumination-adjustable images. However, they can only use one dimension of them because MPEG is tailor-made for movies, which are moving forward or backward in one direction only. But the lighting domain of illumination-adjustable images are 2D in nature. Therefore, MPEG is still unable to fully make use of the data correlation. Thus, without considering the data nature, the compression algorithm cannot effectively reduce the data volume.

1.3 Our Approach

Unfortunately, there have been less work in compressing image-based representation for relighting. Wong *et al.* [1] compressed the relighting data using spherical harmonics. Higher compression ratio is obtained by exploiting the inter-pixel correlation [10]. However, compressing illumination data with spherical harmonics suffers from imprecise reconstruction of highlight and shadows. In this thesis, we propose a compression scheme based on principal component analysis (PCA). There have been some previous work [11, 12, 13, 14] using PCA. While their focuses are mainly on recognition and/or representation, our focus is on the practicality, effectiveness and efficiency of the compression algorithm.

We use a block-based approach to reduce the problem to a manageable size. The output of PCA is a set of principal components and their corresponding coefficients. Most energy of the original data are packed in a few principal components after PCA. Hence, by dropping most of the low-energy components, the data size can be drastically reduced. We then apply transform coding and quantization to further compress the high-energy components. At the same time, the coefficients are compressed using uniform quantization.

The proposed method compresses image-based relighting data to a small size that can be rapidly transferred through Internet. To demonstrate its ability of preserving visual quality, we evaluate the compressed data by measuring the reconstruction error. Moreover, we also consider the capability of real-time relighting from the compressed data.

The work in this thesis has been submitted to several organizations such as [15] and [16].

1.4 Structure of the Thesis

We first give a review on existing compression methods for image-based representation in Chapter 2. Then the image-based relighting representation is briefly described in Chapter 3. From Chapter 4 to Chapter 7, we present the proposed compression method in detail. The relighting is then described in Chapter 8. Overall evaluation is given in Chapter 9. Finally, conclusions are drawn and future directions are discussed in Chapter 10.

Chapter 2

Related Work

In recent years, lots of work in image-based modeling and rendering have been done. However, not most of the work have been devoted to the compression issue of image-based data. We roughly classify the previous work into two main categories, namely compression for navigation and compression for relighting.

2.1 Compression for Navigation

2.1.1 Light Field/Lumigraph

There are several representations proposed for image-based navigation. Levoy and Hanrahan [4] and Gortler *et al.* [5] proposed the light field and lumigraph representations respectively. Both use a two-plane parameterization to represent radiance along ray passing through the enclosed light slab. As pointed out by Chai *et al.* [17], a large number of samples may be required for a smooth view interpolation. Therefore compression is crucial. Levoy and Hanrahan [4] used vector quantization and entropy coding for compression. On the other hand, Gortler *et al.* [5] proposed a transform coding for encoding the data. Utilizing the disparity maps, Magnor and Girod [18, 8] proposed the MPEG-like codec for light field. Zhang and Li [19] proposed a multiple reference frame (MRF) prediction approach for compressing the lumigraph. To progressively

transfer the compressed light field, Peter and Straber [20] used a 4D wavelet decomposition approach for compression. Ihm *et al.* [21] proposed the spherical light field representation. Unlike the two-plane structure in light field and lumigraph, it is spherical in nature. They proposed to use a wavelet-based method for compression. The compression ratio was about the same as the one described in [4]. Bajaj *et al.* [9] proposed a general compression scheme for high dimensional data such as light field. They focused on high compression ratio and fast random access issues. They used wavelet-based transform coding for compression. Indexing and table lookups were heavily used.

2.1.2 Surface Light Field

Miller *et al.* [22] proposed the surface light field, which was a hybrid geometric-based and image-based model. The idea was to allow each surface element to have its own light field. They proposed to use block-based transform coding for compression. Block cache was used to speed up decompression. Wood *et al.*[23] and Chen *et al.*[24] proposed to use principal component analysis and vector quantization for compressing the surface light field.

2.1.3 Concentric Mosaics

Unlike the 4D nature of light field / lumigraph, concentric mosaics proposed by Shum and He [6] was 3D. A standard MPEG4 codec was used to compress these concentric mosaics in their paper. Wu *et al.* [25] proposed a smart-rebinning scheme for concentric mosaics. The idea was to rebin the data before compression in order to expose more correlations among data. Leung and Chen [26] proposed a MPEG-like motion compensation method for compressing the concentric mosaics.

2.1.4 On the Compression

Li *et al.*[27] surveyed three categories of image-based rendering compression algorithms. The block coder was simple and fast in decoding, but the compression ratio was usually not high. The reference block coder could achieve both good compression ratio and fast rendering. Finally, the high dimensional wavelet coder could also achieve high compression ratio and fast rendering. However, the coder was more complex and relatively difficult to implement.

2.2 Compression for Relighting

2.2.1 Previous Approaches

Nimeroff *et al.* [28] used steerable basis function to relight images due to natural illumination. Compression could be achieved by storing coefficients of the basis functions.

Wong *et al.*[1] proposed to treat each pixel as a 'surface element' and measured its apparent bidirectional reflection distribution function (BRDF). By looking up this pixel BRDF, relighting could be performed. They used spherical harmonics [29] to transform the radiance values to frequency domain. The spherical harmonic coefficients were then rebinned and compressed using transform coding [10]. A high compression ratio was achieved. However, due to the global support nature of spherical harmonic transform, sharp shadow and specular highlight were usually smoothened and might not be accurately represented. Our method is better than spherical harmonic transform coding as shown in Chapter 9.

Belhumeur and Kriegman [11] have proven that three basis images are sufficient to span the illumination cone of images containing Lambertian surfaces. Their concern was mainly on the object recognition under various illumination conditions. Epstein *et al.*[12] have empirically shown that 5 to 7 basis

images are enough for representing scene containing specular objects. The basis images can be extracted using principal component analysis. Zhang [13] proposed a hybrid geometric-based and image-based representation model for relighting. He also used principal component analysis to reduce the data dimensionality. Nishino *et al.*[14] proposed the appearance compression scheme. The geometry of the scene was captured by laser range finder while the appearance (texture) of the object was represented by eigen-textures which were extracted by principal component analysis.

2.2.2 Our Approach

Our proposed compression method is designed for image-based relighting. It also employs principal component analysis as a tool to reduce the data dimensionality. Unlike previous work in object recognition and hybrid representation, we treat the image-based relighting as a sampling and reconstruction problem of the plenoptic function. The problem is pure image-based. We make no assumption on the surface types and no restriction on the trajectory of the light source. As the focus of most previous work is not compression, not many attentions have been paid on optimization of compression ratio, practicability, error analysis, random access ability, and real-time decompression. In contrast, we shall address these issues thoroughly in this thesis.

Chapter 3

Image-Based Relighting

For the completeness of this thesis, we give a brief description of the illumination-adjustable image representation previously proposed. The details can be found in [1]. Our image representation is based on the general plenoptic function. We treat the image synthesis as a problem of sampling and reconstruction of the plenoptic function.

3.1 Plenoptic Illumination Function

The original *plenoptic function* [30] describes the radiance received along any viewing direction (θ_v, ϕ_v) arriving at any point (E_x, E_y, E_z) in space, at any time t and over any range of wavelength λ. A similar concept known as *light field* was coined by Gershun [31] to describe the radiometry properties of light in space. The plenoptic function is formulated as follows,

$$I = \mathrm{P}(\theta_v, \phi_v, E_x, E_y, E_z, t, \lambda), \tag{3.1}$$

where I is the radiance, $(E_x,\ E_y,\ E_z)$ or $\dot{E}$ is the position of the center of projection or the viewpoint, $(\theta_v,\ \phi_v)$ or $\vec{V}$ specifies the viewing direction in spherical coordinate, t is the time parameter, and λ is the wavelength.

Basically, the function is formulated to mimic an ideal human eye. It tells us how the environment looks like when our eye is positioned at $\dot{E}$. Figure 3.1

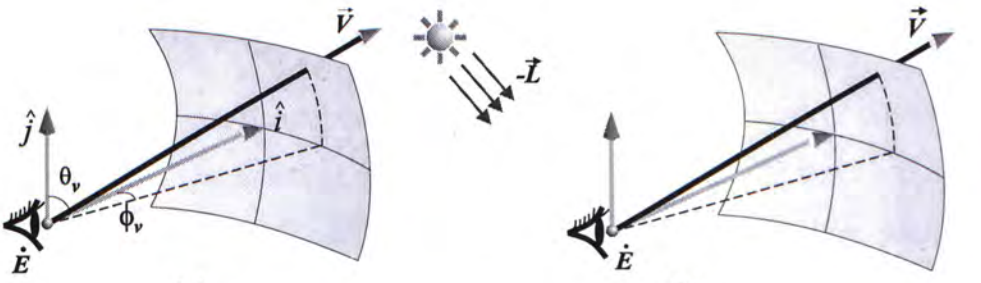


Figure 3.1: Geometric components of the (a) plenoptic and (b) plenoptic illumination functions.

illustrates the geometric components of the function graphically. The time parameter t models all other unmentioned factors such as the change of illumination and the change of scene. When t is constant, the scene is static and the illumination are fixed.

We extended the plenoptic function to include the illumination component [1]. To do so, we extract an illumination component from the aggregate time parameter t and explicitly specify it in the following new formulation,

$$I = \mathrm{P_I}(\theta_l, \phi_l, \theta_v, \phi_v, E_x, E_y, E_z, t', \lambda), \tag{3.2}$$

where (θ_l, ϕ_l) or $\vec{L}$ specifies the direction of a directional light source illuminating the scene, and t' is the time parameter after extracting the illumination component.

We call this new formulation the *plenoptic illumination function*. It describes the lighting direction of a directional light, which emits unit radiance, illuminating the scene. The new function tells us what we see when the whole scene is illuminated by a directional light source with the lighting direction $-\vec{L}$.[1] (Figure 3.1).

[1]We define $\vec{L}$ in local coordinate system.

3.2 Sampling and Relighting

Sampling the plenoptic illumination function is actually a process of taking pictures. The time parameter t' (the scene) is assumed fixed and the wavelength parameter λ is conventionally sampled and reconstructed at three positions (red, blue and green). As we are interested in constant-viewpoint images, parameter $\dot{E}$ is also fixed. Hence the sampling of the viewing direction $\vec{V}$ depends on the projection manifold we used. The manifold can be planar, cylindrical or spherical. We sample the illumination component $\vec{L}$ on the grid points of spherical coordinate system (Figure 3.2).

(a) (b)

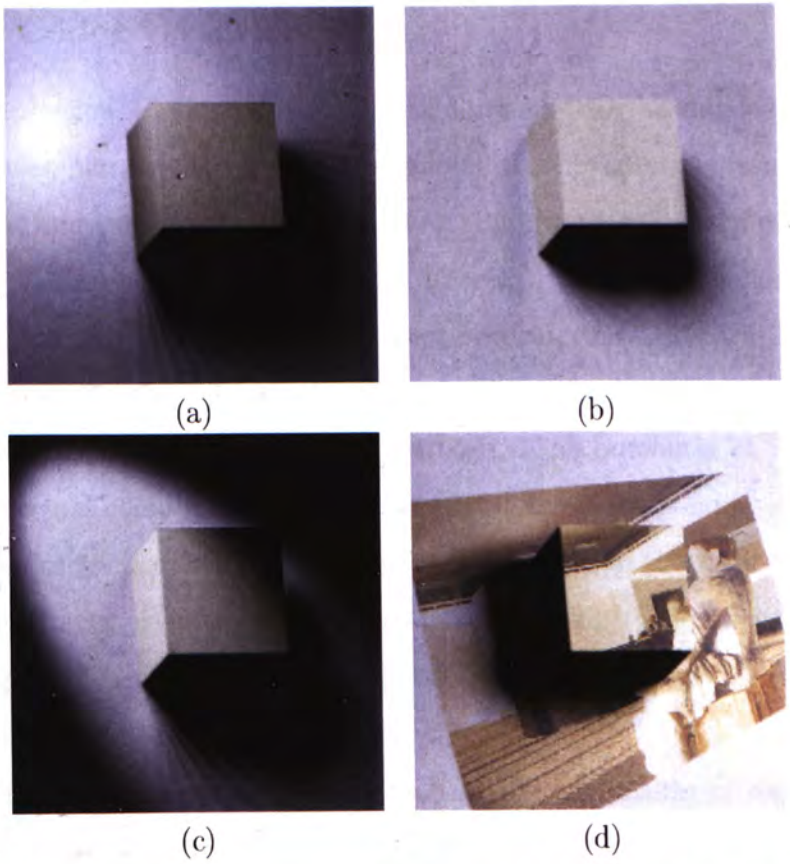
Figure 3.2: (a) Sampling the lighting direction at the grid points on the sphere. (b) A sampled plenoptic function.

Given a desired light vector which is not one of the samples, the desired image of the scene can be estimated by interpolating the samples. We shall discuss the interpolation in Chapter 8. Even though the sampled plenoptic illumination function only tells us how the environment looks like when it is illuminated by a directional light source with unit intensity, other illumination configuration can be simulated by making use of the linearity property of light. The relighting can be done by calculating the following formula for each pixel and for each red, green and blue channel.

$$\sum_{i}^{n} P_I^*(\theta_l^i, \phi_l^i) L_r(\dot{x}, \theta_l^i, \phi_l^i), \tag{3.3}$$

where n is the total number of light sources, (θ_l^i, ϕ_l^i) specifies the desired lighting direction, $\vec{L}_i$, of the i-th light source, $P_I^*(\theta_l^i, \phi_l^i)$ is the result of interpolating the samples given the desired light vector (θ_l^i, ϕ_l^i) (parameters $\vec{V}$, $\dot{E}$, t' and λ are dropped for simplicity), L_r is the radiance along (θ_l^i, ϕ_l^i) due to the i-th light source, and $\dot{x}$ is the position where the actual reflection takes place.

P_I^* is the estimated radiance by interpolating the samples. The above formula is a local illumination model. It allows us to manipulate three parameters, namely the direction, the color, and the number of light source. Figure 3.3 shows the result of using the above illumination model (Equation 3.3) with different types of light sources [1]. Figure 3.4 shows that the same illumination model can be applied to panoramic images [3] as well.



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

Figure 3.3: (a) Point light source, (b) directional light source, (c) spotlight, and (d) slide projector source.

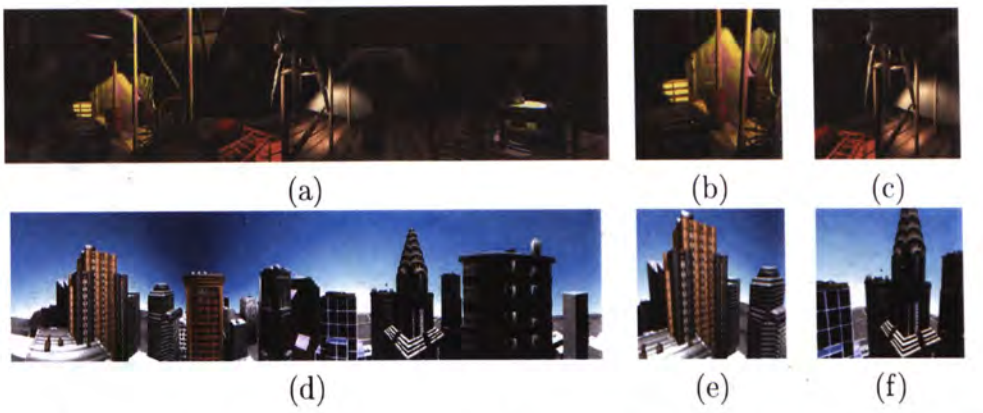


Figure 3.4: Relighting of panorama. (a)-(c): indoor attic scene, and (d)-(f): outdoor city scene.

3.3 Overview

3.3.1 Codec Overview

The compression of illumination-adjustable image can be divided into four main phases, namely *data preparation*, *principal component analysis*, *eigenimage coding* and *relighting coefficient coding*. Figure 3.5(a) illustrates the main phases. The input images (images of different illumination conditions) are color-transformed and divided into 'blocks' during data preparation (Chapter 4). Next, during the principal component analysis (PCA) (Chapter 5), the dimensionality of each 'block' is reduced. The outcomes of PCA are a set of principal components (*eigenimages*) and the corresponding coefficients (*relighting coefficients*). They are separately encoded in the next two phases. The eigenimages are encoded using transform coding (Chapter 6) while the relighting coefficients are uniformly quantized (Chapter 7). Both of them are then further compressed using entropy coding.

On the other hand, the decoding is basically the inverse of the processes (Figure 3.5(b)). However, since we only need to obtain the image of a specific lighting condition, we only decompress a few sets of relighting coefficients each time. Given a desired lighting condition, the corresponding desired relighting

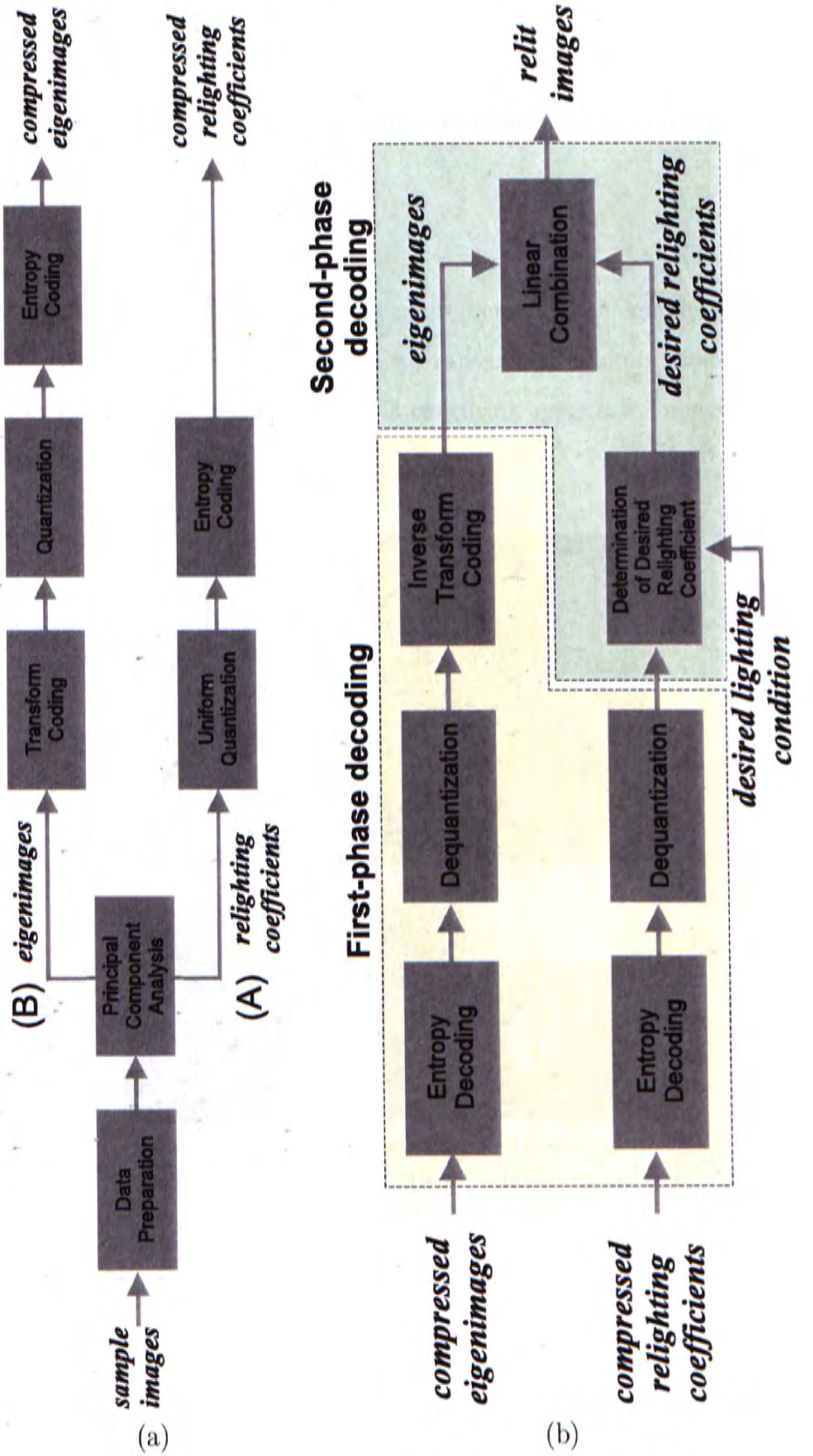
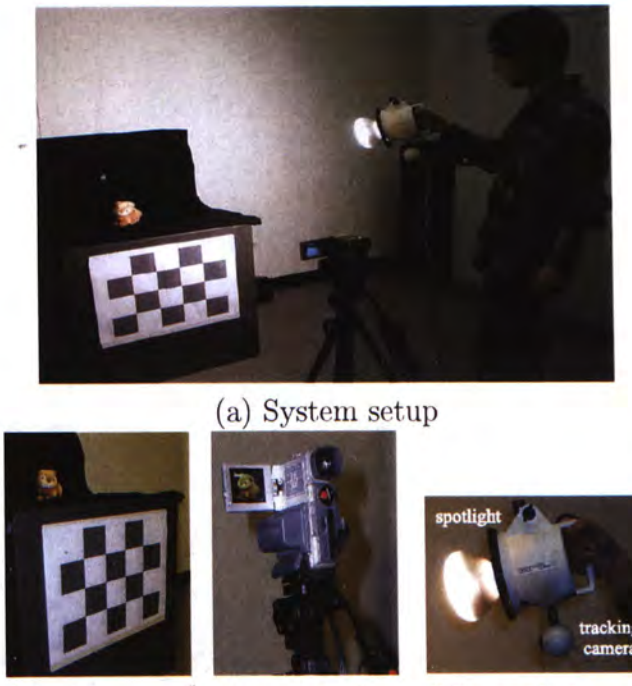


Figure 3.5: Overview of (a) encoding and (b) decoding.

coefficients are interpolated. Then the reconstruction (or *relighting*) can be easily done by linearly combining the eigenimages with the interpolated relighting coefficients as weights. We shall describe in details the relighting in Chapter 8.

3.3.2 Image Acquisition

Although image acquisition is not our main focus, for the completeness of the thesis, we will briefly introduce how we capture the illumination-adjustable images. Pang [32] develops a portable capturing system for image-based relighting. Figure 3.6 shows the system setup.



(a) System setup

(b) object and checker board

(c) capturing camera

(d) spotlight and tracking camera

Figure 3.6: The capturing system setup.

Two cameras are required. The *capturing* camera is for recording the reflected radiance from the object/scene while the *tracking camera* is for tracking

the light vector. The tracking camera is attached to the spotlight, which is the only light source in the system. The checker board helps the tracking camera to locate the current position of the light source. The work flow of the capturing system is shown in Figure 3.7.

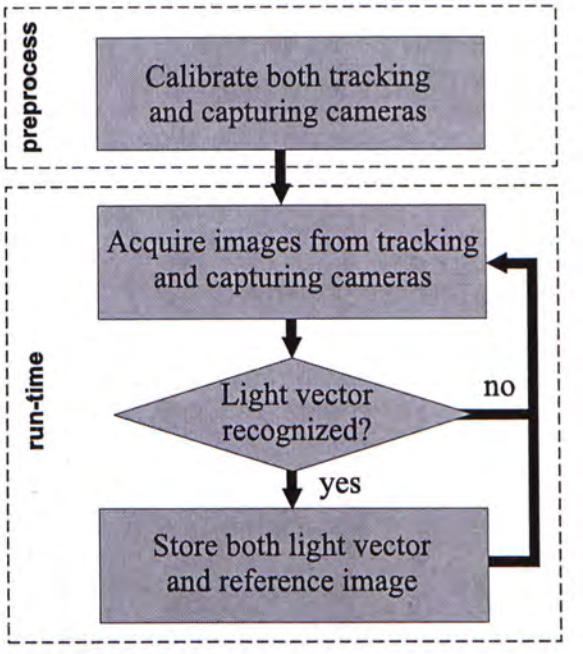


Figure 3.7: Capturing work flow.

The two cameras have to be calibrated beforehand in order to make them distortion-free. During capturing, user can move the spotlight freely. After the light vector has been recognized using the information from the tracking camera, the image from the capturing camera and the light vector are stored. The system keeps running until enough samples have been taken.

3.3.3 Experiment Data Sets

Four data sets (Figure 3.8) are used in our experiments. They are 'ding', 'attic', 'forbidden' and 'cover'. The first three sets are synthetic while the last one contains real-world images captured using the image acquisition system

Data set	**Resolution**	**Sampling rate** $(\theta \times \phi)$	**Real/Synthetic**	**With Shadow**
ding	512×512	30×40	Synthetic	Yes
attic	1024×256	15×20	Synthetic	No
forbidden	1024×256	30×40	Synthetic	Yes
cover	256×256	45×90	Real	Yes

Table 3.1: Characteristics of tested data sets.

described in Section 3.3.2. While the data set 'attic' contains no shadow, all other data sets contain shadow. Table 3.1 shows the characteristics of each data set.

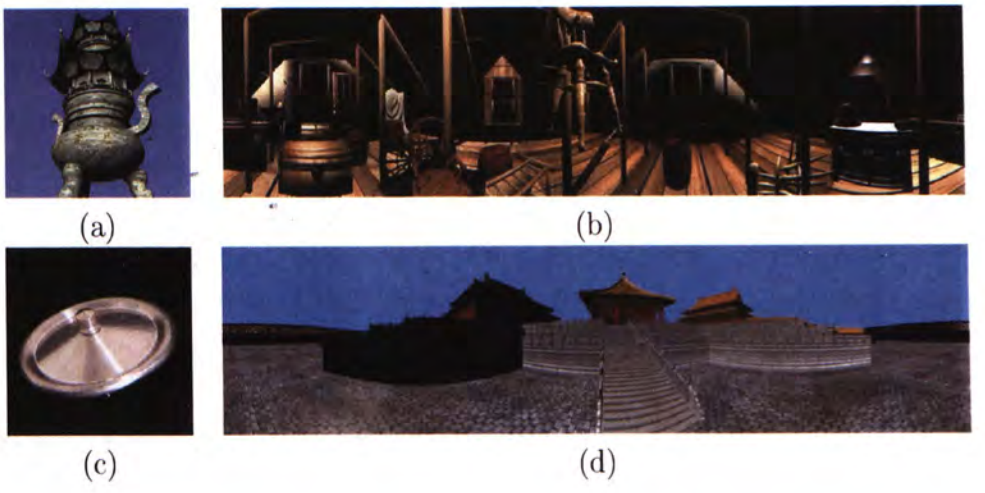


Figure 3.8: Four data sets tested throughout the experiments: (a) 'ding', (b) 'attic', (c) 'cover' and (d) 'forbidden'. Data set 'cover' contains real-world images while others are synthetic.

Chapter 4

Data Preparation

The goal of data preparation is to reorganize (rebin) and preprocess the input data in order to maximize the correlation among neighboring data. So the principal component analysis in the following stage can effectively reduce the data dimensionality. There are three major steps, namely block division, color transform, and mean extraction. Block division aims at breaking down the data in order to facilitate and speed up the principal component analysis process. By transforming the data to another color domain, we try to make use the human inability in order to improve the performance (higher compression ratio with better reconstruction quality). Mean extraction tries to zero mean the input data in order to optimize the data for principal component analysis.

4.1 Block Division

The input is a set of images with the same view, but with different illumination condition. Figures 4.1, 4.2, 4.3, and 4.4 show three samples from each of our data sets, 'attic', 'ding', 'forbidden', and 'cover', each illuminated by a single directional light source from different direction. Note that shadow is present in the sample images. We sample the direction of light source on a spherical grid as illustrated on the left hand side of Figure 4.5. Hence each image corresponds to a grid point. The image set is indexed by (θ_i, ϕ_j) for $i = 0, \cdots, p-1$ and

Figure 4.1: Three samples from the data set 'attic', each illuminated with different lighting condition.

Figure 4.2: Three samples from the data set 'ding'.

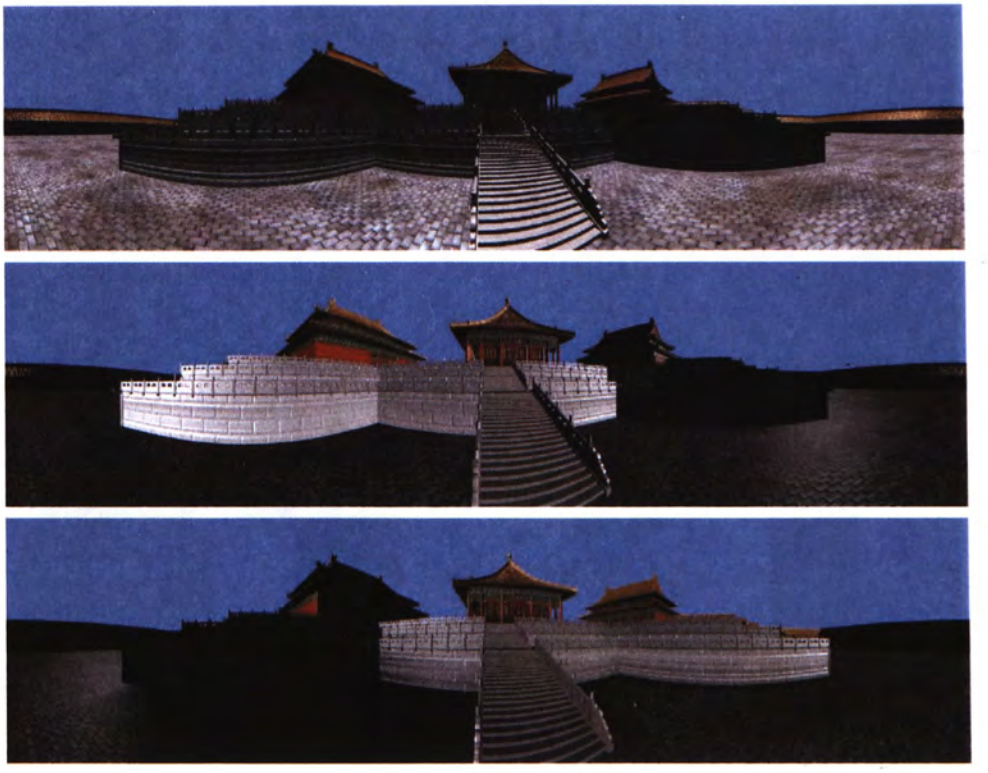


Figure 4.3: Three samples from the data set 'forbidden'.

Figure 4.4: Three samples from the data set 'cover'.

$j = 0, \cdots, q-1$. The set contains altogether $m = pq$ images.

If we select a pixel in the image (as in Figure 4.5) and pick all values associated with this pixel, it is not surprising that these values are highly correlated. Because these values are the radiances reflected from the same surface element visible through the pixel window under various illumination conditions. In other words, the variation in reflection is mainly due to the bidirectional reflection distribution function (BRDF) of that surface element.

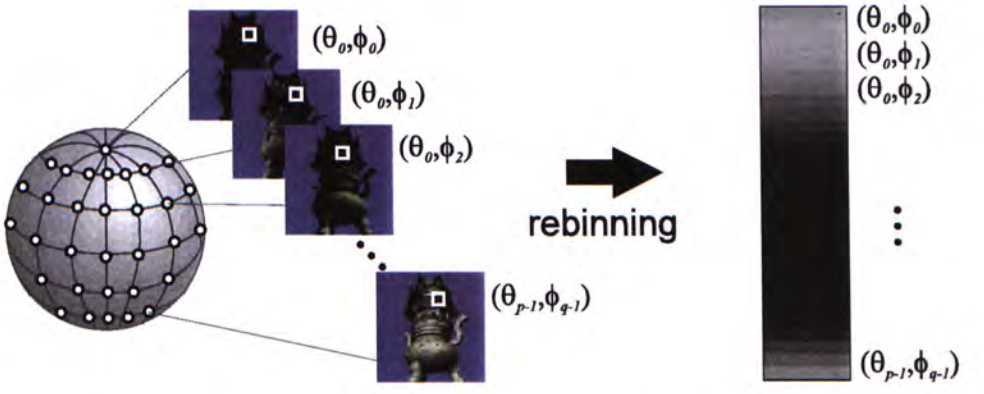


Figure 4.5: Strong correlation is observed in the pixel values among images.

Since the input images are highly correlated, we apply PCA to bring down the data dimensionality. One can linearize each 2D image (probably a 2D array of pixel values) to a 1D array of pixel values. Then all 1D arrays (images) are stacked to form a data matrix **M**. However, this matrix is prohibitively large for computation. For example, a gray scale image of 256 × 256 sampled under m=1200 lighting conditions forms a 1200 × 65535 data matrix. The computation of the principal components of this matrix is impractical.

We propose a divide-and-conquer approach which subdivides the images into blocks. Multiple *block-wise PCAs* are then applied on the corresponding blocks. Spatially neighboring blocks are handled independently. If each image is subdivided into w blocks, we perform w block-wise PCAs and each on different set of blocks as illustrated in Figure 4.6. With this block-based

approach, the computation becomes tractable and the memory requirement is also reduced. Moreover, the computation can be trivially parallelized.

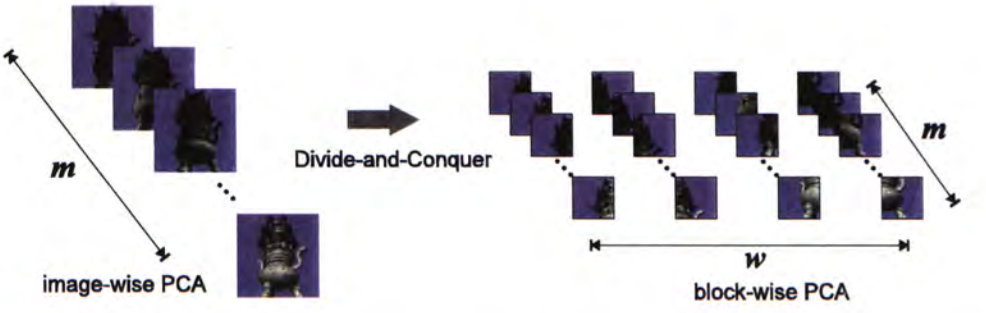


Figure 4.6: A divide-and-conquer approach is used to make the computation tractable and introduce parallelism.

Since spatially neighboring blocks are handled independently, block-wise PCAs can be trivially parallelized. The block-wise PCA also helps in capturing high-frequency features, like highlight and shadows, with fewer number of principal components. However, the subdivision leads to overhead which in turn reduces the compression ratio. We shall discuss the overhead shortly. In general, the smaller the block is, the higher the parallelism is, but the lower the compression ratio is.

In our system, we choose a block size of 16×16. Although the block size of 8×8 is widely used in transform coding like JPEG [33], there is too much storage overhead. Since we employ transform coding in the later stage, it is desirable to keep the block size to be multiple of 8×8 in order to avoid any artifacts due to the boundaries. On the other hand, a block size of 32×32 (or above) brings up the size of the matrix, making the computation expensive. This is not desirable. However, a block size of 16×16 introduces less overhead while maintaining the compatibility with standard transform codec. A similar strategy is used in designing the size of macroblock in MPEG standard [34].

4.2 Color Model

In our applications, we handle color images. By converting the pixel values from RGB to YC_rC_b color model, we separate the luminous component (Y) from the chrominuous components (C_r and C_b). Hence we can allocate more bits for human-sensitive luminous component while less bits for human-insensitive chrominuous components during compression in the later stage. We use the following two matrices [35] for forward and backward transforms between color models.

The forward transform, from RGB to YC_rC_b,

$$\begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} = \begin{bmatrix} 0.2990 & 0.5870 & 0.1140 \\ 0.5000 & -0.4187 & -0.0813 \\ -0.1687 & -0.3313 & 0.5000 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4.1}$$

The backward transform, from YC_rC_b to RGB,

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1.40200 & 0.00000 \\ 1 & -0.71414 & -0.34414 \\ 1 & 0.00000 & 1.77200 \end{bmatrix} \begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix} \tag{4.2}$$

The range of the original RGB pixel values is [0,1] (we map pixel values from [0,255] to [0,1]). The corresponding ranges of channels Y, C_r, and C_b are [0, 1], [-0.5, 0.5], and [-0.5, 0.5] respectively.

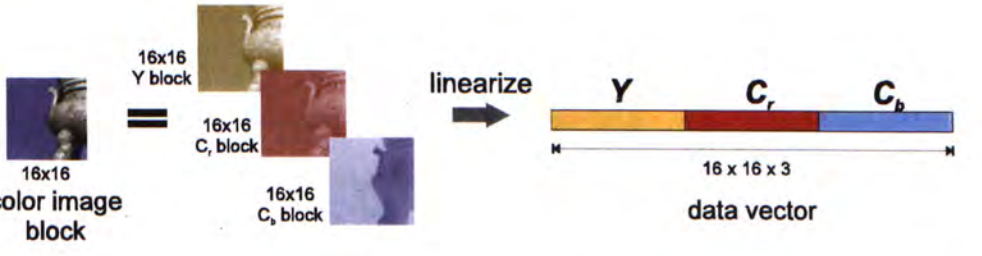


Figure 4.7: A color image block is linearized to form a data vector. Its Y, C_r and C_b data are grouped inside the same vector.

Since there exists high correlation between values in the three channels (YC_rC_b), we group values from the three channels and perform PCA on them as a whole (mixed channel PCA). Figure 4.7 illustrates how a color image block is linearized and forms a data vector. On the other hand, if we perform PCA separately on each channel (separated channel PCA), the storage requirement will increase (as each PCA generates additional set of coefficients and principal components). From our experiments, we found that mixed channel PCA not just reduces storage overhead, but also preserves image quality. Figure 4.8 compares the reconstruction errors between mixed channel PCA and separated channel PCA. The horizontal axis indicates the number of eigenimages used to reconstruct the image. As the number of eigenimage increases, the reconstruction error decreases. The error due to mixed channel PCA is almost the same as the error level due to separated channel PCA when more than four eigenimages are used for reconstruction. This phenomenon is consistent in different data sets.

4.3 Mean Extraction

It is beneficial to perform mean extraction on the data matrix in prior to PCA. Just like the mean extraction performed in the previously proposed SVD-based still image coders such as [36]. The mean extraction, also known as rank-one update, shifts the centroid of the data cloud to origin and usually allows more precise reconstruction in compare to that without mean extraction. it is believed that this is more likely to provide higher compression ratio given the same error tolerance.

Unlike previous approaches [14], we compute the mean image (mean vector) over m input images instead of a scalar mean value. The mean vector offers a more accurate computational base for PCA. Each input image is subtracted by the mean image before forming the data matrix for PCA. During image

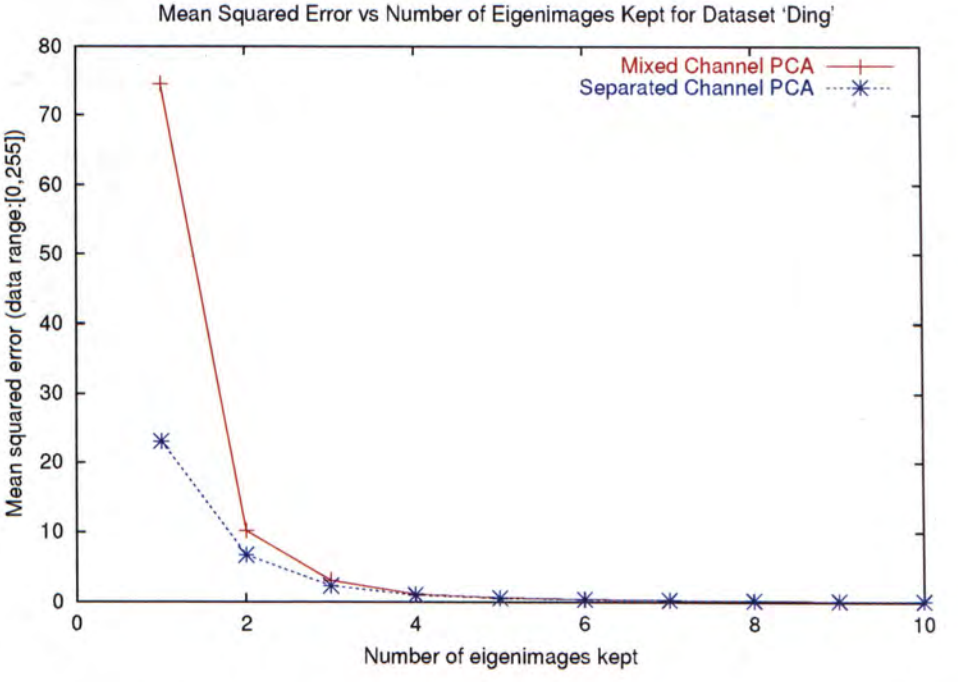
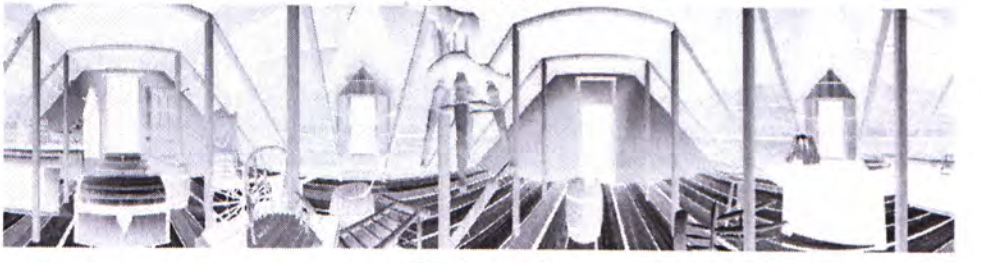


Figure 4.8: Error comparison (MSE) between PCA on grouped data and PCA on separated channel data. Data set 'ding' is used in this experiment.

reconstruction, the mean image is added to recover the images. Since we perform block-wise PCA, we compute w mean image blocks instead of a single mean image. The mean corrected data is now of range $[-1, 1]$. Figures 4.9, 4.10, 4.11, and 4.12 show the computed mean image blocks (we tile the mean image blocks) of all the data sets. The images are all stretched to [0..255] in order to allow a better visualization. The original images of the chrominuous channels should look much dimmer.

Y channel

C_r channel



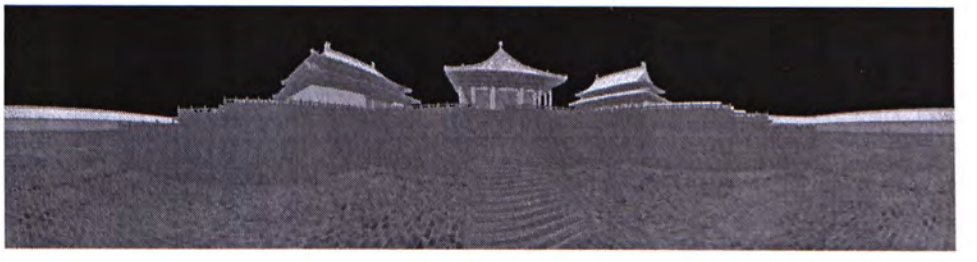
C_b channel

Figure 4.9: The mean image blocks of data set 'attic'. The mean image blocks are tiled according to their original position in the image. Note that all images are scaled to [0..255] in order to increase the contrast for visualization. The images for C_r and C_b are dimmer originally.

Y channel C_r channel C_b channel

Figure 4.10: The mean image blocks of data set 'ding'.

Y channel



C_r channel

C_b channel

Figure 4.11: The mean image blocks of data set 'forbidden'.

Y channel C_r channel C_b channel

Figure 4.12: The mean image blocks of data set 'cover'.

Although there are w mean image blocks, you cannot figure out any artifacts at the boundaries. It is because the mean of each set of corresponding pixels in the data set is independent of the block size.

Chapter 5

Principal Component Analysis

After block dividing, color transforming, and mean extracting the input data, we attempt to reduce the data volume in this phase. Principal component analysis, more specifically, singular value decomposition, is used. All aspects concerning singular value decomposition that are related to our work, such as how singular value decomposition can reduce the data volume, will be discussed. At the end of this chapter, we give an evaluation on how the performance of singular value decomposition is in reducing data volume.

5.1 Overview

After data preparation, all data vectors are stacked to form a data matrix $\mathbf{M}$ and block-wise PCA is applied to $\mathbf{M}$. Our goal is to reduce the dimensionality of input data (m data vectors). The output of PCA can be well approximated by k basis images (*eigenimages*) and their corresponding coefficients (*relighting coefficients*). Since $k \ll m$, the data volume is drastically reduced by keeping only k eigenimages and the relighting coefficients.

Figure 5.1 illustrates this dimensionality reduction process graphically. Each row of data matrix $\mathbf{M}$ corresponds to one input image block. Hence each row contains $n = 16 \times 16 \times 3$ scalars. Through PCA, we can decompose the data matrix into two matrices, $\mathbf{A}$ and $\mathbf{B}$. The beauty of PCA is that most

energy of data matrix **M** concentrates in the first few components of the decomposed matrices (first few columns of **A** and first few rows of **B**). In other words, by keeping only the first k components (*i.e.* keeping the first k columns of **A** and first k rows of **B**), we can preserve most energy of the input data and achieve compression.

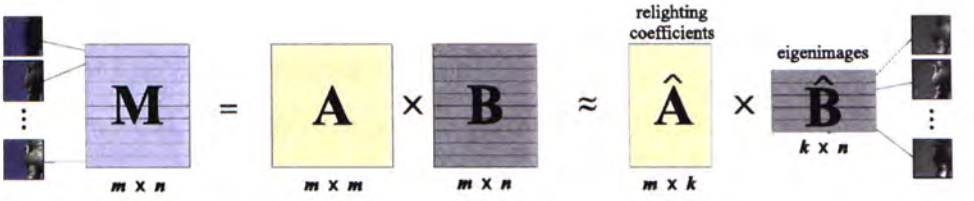


Figure 5.1: Using PCA, we approximate the original data matrix (**M**) by two smaller matrices $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{B}}$.

5.2 Singular Value Decomposition

Singular value decomposition and its development was mainly established by five mathematicians: Eugenio Beltrami (1835-1899), Camille Jordan (1838-1921), James Joseph Sylvester (1814-1897), Erchard Schmidt (1876-1959), and Hermann Weyl (1885-1955). For the history of SVD, interested parties can refer to [37].

We use singular value decomposition (SVD) [38, 37] as the PCA tool to extract the principal components. SVD factorizes an $m \times n$ matrix **M** into 3 matrices,

$$\mathbf{M} = \mathbf{USV}^T \tag{5.1}$$

where matrices **U** and $\mathbf{V}^T$ are of dimensions $m \times m$ and $m \times n$ respectively. They are both unitary matrices. A unitary matrix, say **U**, guarantees that

$$\mathbf{U}^H\mathbf{U} = \mathbf{I}$$

where $\mathbf{U}^H$, which is always equal to $\mathbf{U}^{-1}$, denotes the adjoint matrix of $\mathbf{U}$ and $\mathbf{I}$ is an identity matrix. In our case, the adjoint matrices $\mathbf{U}^H$ and $(\mathbf{V}^T)^H$ are equal to $\mathbf{U}^T$ and $(\mathbf{V}^T)^T$ respectively. The columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{M}$ while the rows of $\mathbf{V}^T$ are called the right singular vectors of $\mathbf{M}$. Matrix $\mathbf{S}$ is a diagonal matrix of dimension $m \times m$. Its elements are the singular values of $\mathbf{M}$. These elements of $\mathbf{S}$ are conventionally sorted in descending order, such that,

$$s_0 \geq s_1 \geq s_2 \geq \cdots \geq s_{m-1} \geq 0$$

Note that the singular values are non-negative.

In practical, the algorithms for SVD assume that m is always bigger than or equal to n. We can always satisfy this condition by transposing the matrix $\mathbf{M}$ whenever $n > m$,

$$\begin{aligned} \mathbf{M} &= \mathbf{USV}^T \\ \mathbf{M}^T &= (\mathbf{USV}^T)^T \\ &= \mathbf{VSU}^T \end{aligned}$$

Because $\mathbf{S}$ is a diagonal square matrix, $\mathbf{S}$ is equal to $\mathbf{S}^T$. As $\mathbf{V}$ and $\mathbf{U}^T$ are unitary matrices by definition and $\mathbf{S}$ is a diagonal matrix, so $\mathbf{VSU}^T$ is a result of SVD of $\mathbf{M}^T$. However, the dimensions of the matrices seem to be having problems. According to the definition, we expect the dimensions of $\mathbf{V}$, $\mathbf{S}$, and $\mathbf{U}^T$ to be $n \times n$, $n \times n$, and $n \times m$ respectively. But, we have now the dimensions $n \times m$, $m \times m$, and $m \times m$. This can be explained by inspecting the rank of $\mathbf{M}$.

By definition, rank of a matrix is the number of non-zero singular values of it and is bounded by $min(m, n)$. In other words, there are some zero singular

values in $\mathbf{S}$ if it is of dimension $m \times m$ with $m \geq n$,

$$\mathbf{S} = \begin{bmatrix} s_0 & & & & & \\ & \ddots & & & & \\ & & s_{n-1} & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{bmatrix}$$

Therefore, the diagonal square matrix $\mathbf{S}$ can be 'downsized' to $n \times n$ by removing the zero singular values. It is also not difficult to conclude that $\mathbf{V}$ and $\mathbf{U}^T$ can be resized to $n \times n$ and $n \times m$. Hence, whenever $m < n$ (number of samples is fewer than total number of pixels in the blocks of all three channels), we can safely transpose $\mathbf{M}$ and apply the SVD routine on $\mathbf{M}^T$. The matrices $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}$ for $\mathbf{M}$ can then be obtained. For the rest of the thesis, we assume $m \geq n$ unless otherwise specified.

The singular values in $\mathbf{S}$ may be large in magnitude while the values in $\mathbf{U}$ and $\mathbf{V}^T$ are relatively small. More specifically, the singular values in $\mathbf{S}$ is bounded by $[0, \sqrt{mn}]$ while the values in $\mathbf{U}$ and $\mathbf{V}$ are bounded by $[-1, 1]$ (note that $\mathbf{M}$ is of range $[-1, 1]$). For a color data set with 1200 samples using block size of 16×16, the singular values are bounded by $[0, 960]$. Storage and manipulation of both large and small values may harm the accuracy (*e.g.* when a small floating-point value is divided by a large floating-point value). It is also not desirable to compress data source containing both large and small values.

By splitting the singular values into two halves (taking square roots) and multiply them to $\mathbf{U}$ and $\mathbf{V}^T$, we evenly distribute the energy of singular values

to obtain an $m \times m$ matrix $\mathbf{A}$ and an $m \times n$ matrix $\mathbf{B}$.

$$\mathbf{A} = \mathbf{U} \times \begin{bmatrix} \sqrt{s_0} & 0 & \dots & 0 \\ 0 & \sqrt{s_1} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \sqrt{s_{m-1}} \end{bmatrix} \tag{5.2}$$

$$\mathbf{B} = \begin{bmatrix} \sqrt{s_0} & 0 & \dots & 0 \\ 0 & \sqrt{s_1} & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & \sqrt{s_{m-1}} \end{bmatrix} \times \mathbf{V}^T \tag{5.3}$$

In the mean time, we do not need to store any values of $\mathbf{S}$. From our experimental data, we find that most elements in $\mathbf{A}$ and $\mathbf{B}$ fall into the range of $[-1, 1]$. This nice property facilitates our hardware-assisted relighting in Chapter 8. However there are rare cases that some values are out of this range. For example, if

$$\mathbf{M} = \begin{bmatrix} -0.3 & 1.0 \\ -0.1 & -0.1 \\ -0.2 & -0.3 \end{bmatrix}$$

The mean of $\mathbf{M}$ is zero. Applying SVD on it, we get

$$\mathbf{U} = \begin{bmatrix} 0.9704 & 0.2396 \\ -0.0702 & 0.4040 \\ -0.2312 & 0.8828 \end{bmatrix} \mathbf{S} = \begin{bmatrix} 1.0734 & 0 \\ 0 & 0.2962 \end{bmatrix} \mathbf{V} = \begin{bmatrix} -0.2216 & -0.9751 \\ 0.9751 & -0.2216 \end{bmatrix}$$

By splitting $\mathbf{S}$ into two halves, we get

$$\mathbf{US}^{0.5} = \begin{bmatrix} 1.0054 & 0.1304 \\ -0.0727 & 0.2199 \\ -0.2395 & 0.4805 \end{bmatrix} \qquad \mathbf{S}^{0.5}\mathbf{V}^T = \begin{bmatrix} -0.2296 & 1.0103 \\ -0.5307 & -0.1206 \end{bmatrix}$$

There are two elements in $\mathbf{US}^{0.5}$ and $\mathbf{S}^{0.5}\mathbf{V}^T$ that exceed the range [-1,1]. The visual artifacts due to this during hardware-assisted relighting (there is no harm to software relighting as the precision problem is negligible) will be discussed in Chapter 8.

By splitting $\mathbf{S}$ evenly into two halves, the matrix $\mathbf{M}$ is now rewritten as

$$\mathbf{M} = \mathbf{AB} \tag{5.4}$$

Every row of $\mathbf{M}$ (image block) can be reconstructed by linearly combining all rows in $\mathbf{B}$. The corresponding weights (coefficients) are kept in a row in $\mathbf{A}$. Figure 5.2 illustrates this reconstruction graphically. We call the rows in $\mathbf{B}$ the basis images or *eigenimages* and the weights in $\mathbf{A}$ the *relighting coefficients*.

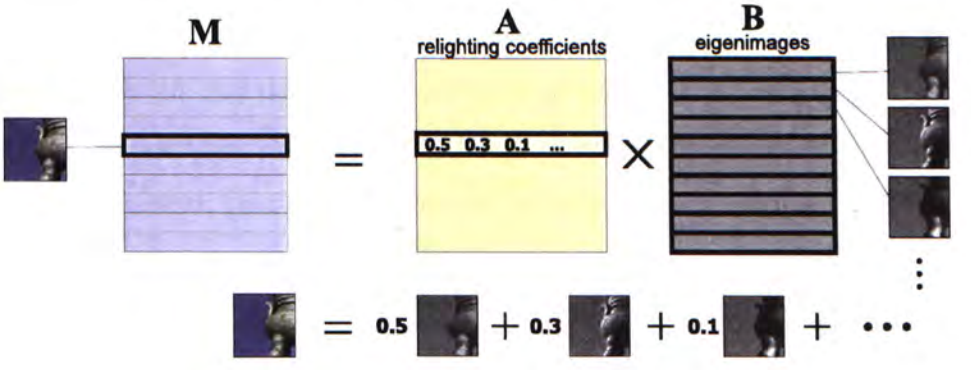


Figure 5.2: Each row (image block) of $\mathbf{M}$ is a linear combination of rows (eigenimages) in $\mathbf{B}$.

5.3 Dimensionality Reduction

SVD packs the energy to a few components. Figure 5.3 shows that the singular value drops rapidly from the first to the fourth singular values. Hence, the image blocks can be well approximated by linearly combining only the first few eigenimages. In this way, the matrix $\mathbf{M}$ can be approximated by

$$\mathbf{M} = \mathbf{AB} \approx \hat{\mathbf{A}}\hat{\mathbf{B}} \tag{5.5}$$

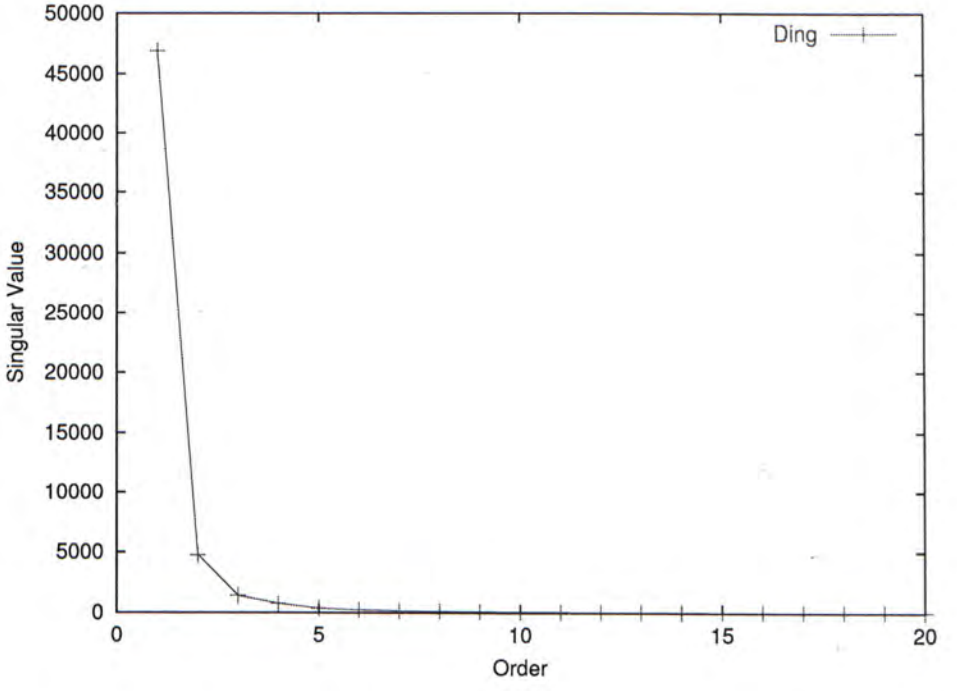


Figure 5.3: The singular value decreases rapidly (data set: 'ding').

where the $m \times k$ matrix $\widehat{\mathbf{A}}$ is formed by keeping only the first k columns of $\mathbf{A}$ and the $k \times n$ matrix $\widehat{\mathbf{B}}$ is formed by keeping the first k rows of $\mathbf{B}$ (Figure 5.1).

Given an error tolerance, we can determine the number of eigenimages required for approximation. Different error metrics have been proposed. We use the Frobenius norm of the differences between the original and the reconstructed matrices as our error metric. It is the squared truncation error of keeping the first k eigenimages, e_k, which is defined as

$$e_k = \|\mathbf{M} - \widehat{\mathbf{A}}\widehat{\mathbf{B}}\|_F^2 = \sum_{i=k}^{m-1} s_i^2 \tag{5.6}$$

Error e_k is also the sum of squared singular values corresponding to the dropped components. Dividing e_k by the number of elements in $\mathbf{M}$, we obtain the mean-squared-error (MSE) between the original and the approximation. Hence the user can specify an error tolerance in term of MSE and the number of eigenimages required can then be determined.

Here, we include two more error metrics for reference. Zhang [13] proposed the information loss

$$I_{loss}(k) = \sqrt{\frac{\sum_{i=k}^{m-1} s_i^2}{\sum_{i=0}^{m-1} s_i^2}} \tag{5.7}$$

From the equation, we know that the lower the information loss is, the better the quality is. Nishino *et al.*proposed the eigenratio, which was defined as

$$\frac{\sum_{i=0}^{k-1} s_i^2}{\sum_{i=0}^{m-1} s_i^2} \tag{5.8}$$

From the equation, we know that the higher the eigenratio is, the better the quality is. Both metrics depend on the Frobenius norm, which we use as our error metric.

There are plenty SVD algorithms in the literature. The common one can be found in [39]. But in practical, it is not efficient nor effective. The algorithm is iteration-based. It refines all singular vectors and singular values in each iteration. Therefore, the worst case is that we may wait until the last iteration for the result. However, we are only interested in a few singular vectors. In our experiment, keeping 9 singular vectors is enough (Section 5.4). However, the total number of singular vectors is m, which is the number of samples. For the data set 'ding', we have 1200 samples. It is obviously that computing all 1200 singular vectors is not cost effective if we are only interested in 9 singular vectors. Moreover, the time complexity of the algorithm is $O(mn^2)$.

Fortunately, there are other iteration methods for computing only a few desired singular vectors and singular values. Berry *et al.* implemented Lanczos and subspace iteration-based methods for computing the first few largest singular vectors and singular values in their SVDPACKC package [40]. From their experiments, they found that the block Lanczos method outperformed other methods, say subspace iteration, trace minimization, and single-vector Lanczos. Therefore, it is much more cost effective to use block Lanczos method in our system as we are only interested in the first few largest singular vectors and singular values.

5.4 Evaluation

As mentioned before, the radiance values are mainly due to the underlying BRDF of the visible surface element. The ability of PCA to reduce the dimensionality highly depends on the BRDF and geometry. If the surface is Lambertian, Belhumeur and Kriegman [11] have proven that three eigenimages are sufficient to approximate. For real-world specular surfaces, Epstein *et al.* [12] empirically showed that 5 to 7 dimensions were sufficient for approximation. However, they are only some lower boundaries for the number of eigenimages we should use. Most previous work assumed shadowing was absent. However, shadow due to nearby geometry is unavoidable in real world. This shadow may increase the dimensionality for representation. In our work, we make no assumption on the surface reflectance, shadowing, and geometry. The number of eigenimages depends on the required accuracy.

To determine the number of required eigenimages, we plot a graph of reconstruction error versus the number of eigenimages. Instead of MSE, we measure the reconstruction error using peak signal-to-noise ratio (PSNR). The reconstruction errors are calculated using Equation 5.6. Since we calculate the errors directly from the singular values, the errors may not be accurate as the truncation errors due to the precision of data type are avoided. However, we believe that the overall shape of the result is not affected. From Figure 5.4, we find that 9 eigenimages give more than 46 dB which is sufficient for representation even shadow is present. Therefore, rather than use 7 eigenimages, we add 2 more in order to perserve more useful information (and thus a better reconstruction quality). It is because, unlike the previous work, we will further process the output data in a lossy way, we must keep a higher quality at this moment. Hence, we use 9 eigenimages as default in our experiments.

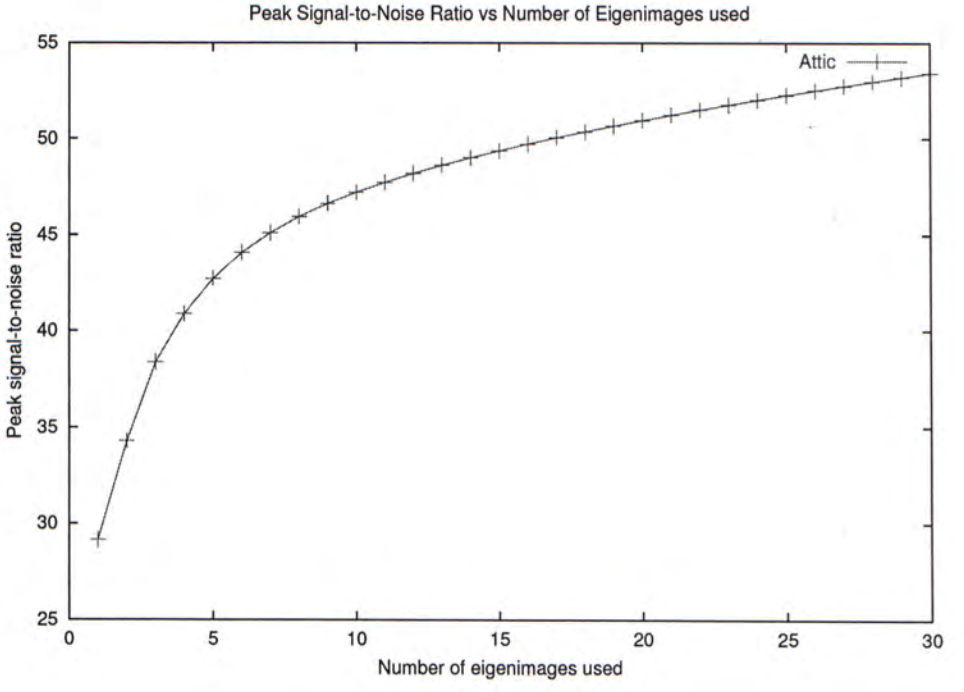


Figure 5.4: PSNR vs number of eigenimages used for reconstruction.

Chapter 6

Eigenimage Coding

The compression ratio is unsatisfactory if we only use PCA to reduce the data volume. In this chapter and next chapter, we discuss how we can further compress the data after applying PCA. In this chapter, we discuss how to compress the eigenimages using transform coding methods as they are images in nature. Two transform coding methods, namely discrete cosine transform and discrete wavelet transform, will be discussed. The performance of using these two different transform coding methods will be given in the end of this chapter, at where we shall see that discrete wavelet transform performs better than discrete cosine transform does. We will also suggest the optimal target bit rates based on our experiment results.

6.1 Transform Coding

After PCA, we obtain a set of eigenimages (matrix $\widehat{\mathbf{B}}$) and the corresponding relighting coefficients (matrix $\widehat{\mathbf{A}}$). Since the properties of these two sets of values are different, we compress them differently. We first discuss the compression of $\widehat{\mathbf{B}}_i$ for $i = 0, \cdots, w-1$. The subscript i denotes which block we are referring as we divide the image into blocks.

In fact, each eigenimage is itself an image. Figures 6.1, 6.2, 6.3, and 6.4 shows the first 6 eigenimages of the Y channel from four tested data sets. Note

that each image is scaled to the range [0,255]. The j-th row in $\widehat{\mathbf{B}}_i$ gives three 16×16 blocks (Y, C_r and C_b blocks) in the j-th eigenimage. We then tile the blocks according to their positions in the original image and form three tiled eigenimages. Figure 6.5 illustrates such tiling. Note that we separate the Y, C_r, and C_b during tiling. Therefore we have altogether $3k$ tiled eigenimages after tiling. Each tiled eigenimage is in the same resolution as the original input images. The tiled eigenimages are denoted as E_j^c for $c = Y, C_r$, or C_b and $j = 0, \cdots, k-1$.

Since we perform block-wise PCA, it is not surprising that boundary is observable in the tiled eigenimages. It seems that low-order eigenimages are smoother than high-order eigenimages. As natural images mainly contain low-frequency signal, it is natural to have *low frequency* low-order eigenimages and *high frequency* high-order eigenimages. In this sense, eigenimages are somewhat analogous to Fourier-transformed images. We compress the tiled eigenimages using transform coding methods. Two transform coding methods, the discrete cosine transform (DCT) and the discrete wavelet transform (DWT), have been tested. Discrete cosine transform is used due to its effectiveness in compressing still images as proven by the popularity of its usage in image coding nowadays. However, the new standard, JPEG 2000, has a better performance, especially in low bit rates, in compressing still images comparing to JPEG. Discrete wavelet transform is used in JPEG 2000, therefore we also try to compress the eigenimages using discrete wavelet transform in this phase.

6.1.1 Discrete Cosine Transform

We use the 2-D discrete cosine transform [41]. The forward transform is

$$C(u,v) = \alpha(u)\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos \frac{(2x+1)u\pi}{2N} \cos \frac{(2y+1)v\pi}{2N} \qquad (6.1)$$

Figure 6.1: The first 6 tiled eigenimages (Y channel) of 'attic'.

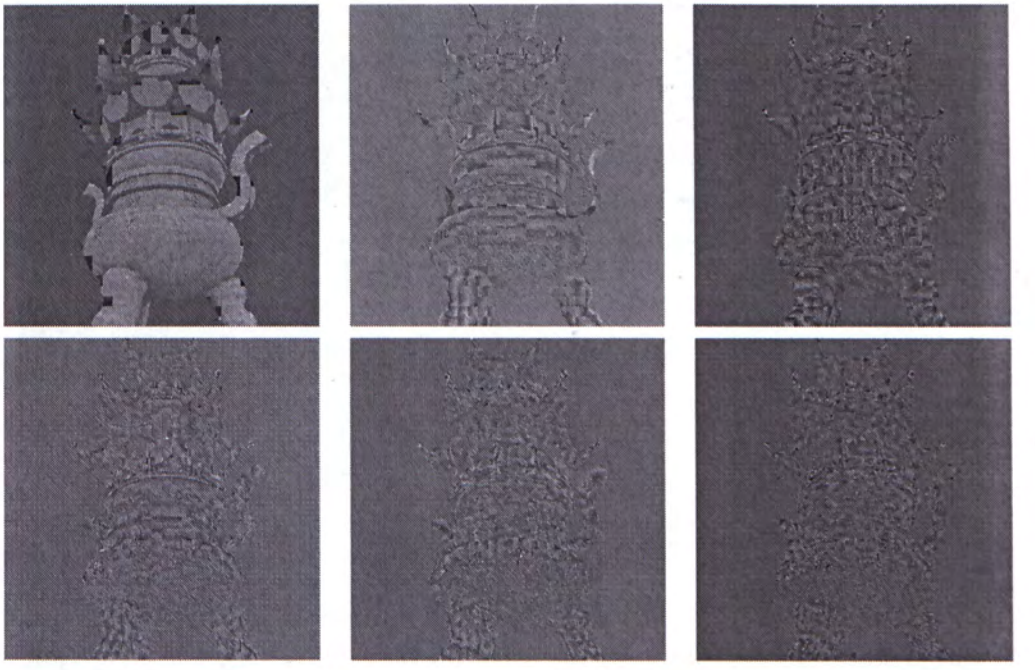


Figure 6.2: The first 6 tiled eigenimages (Y channel) of 'ding'.

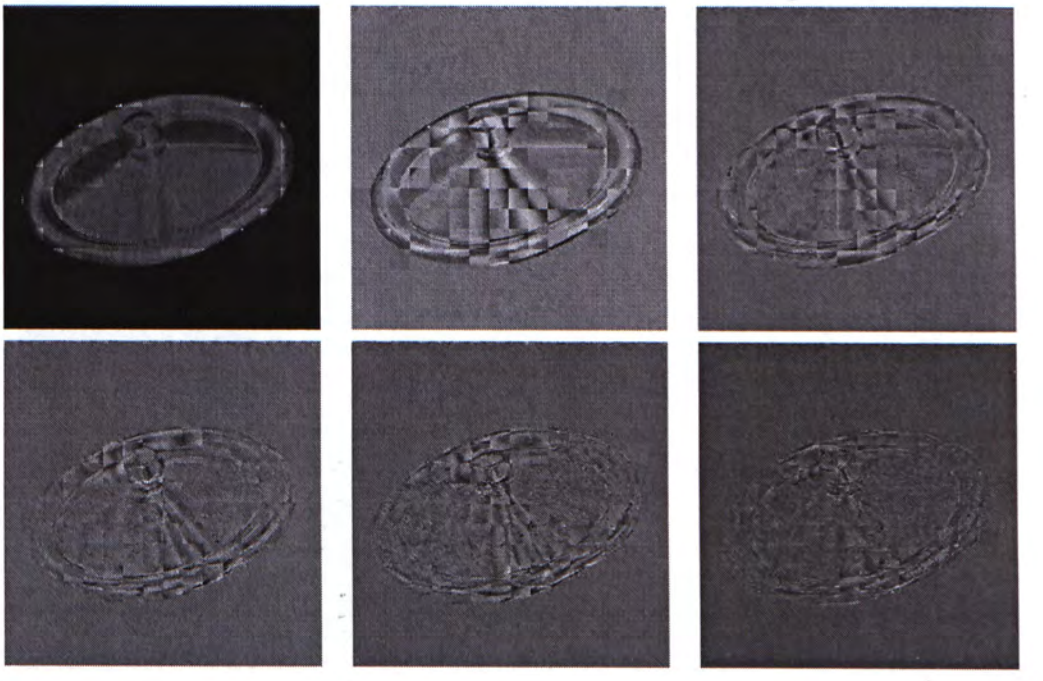


Figure 6.3: The first 6 tiled eigenimages (Y channel) of 'cover'.

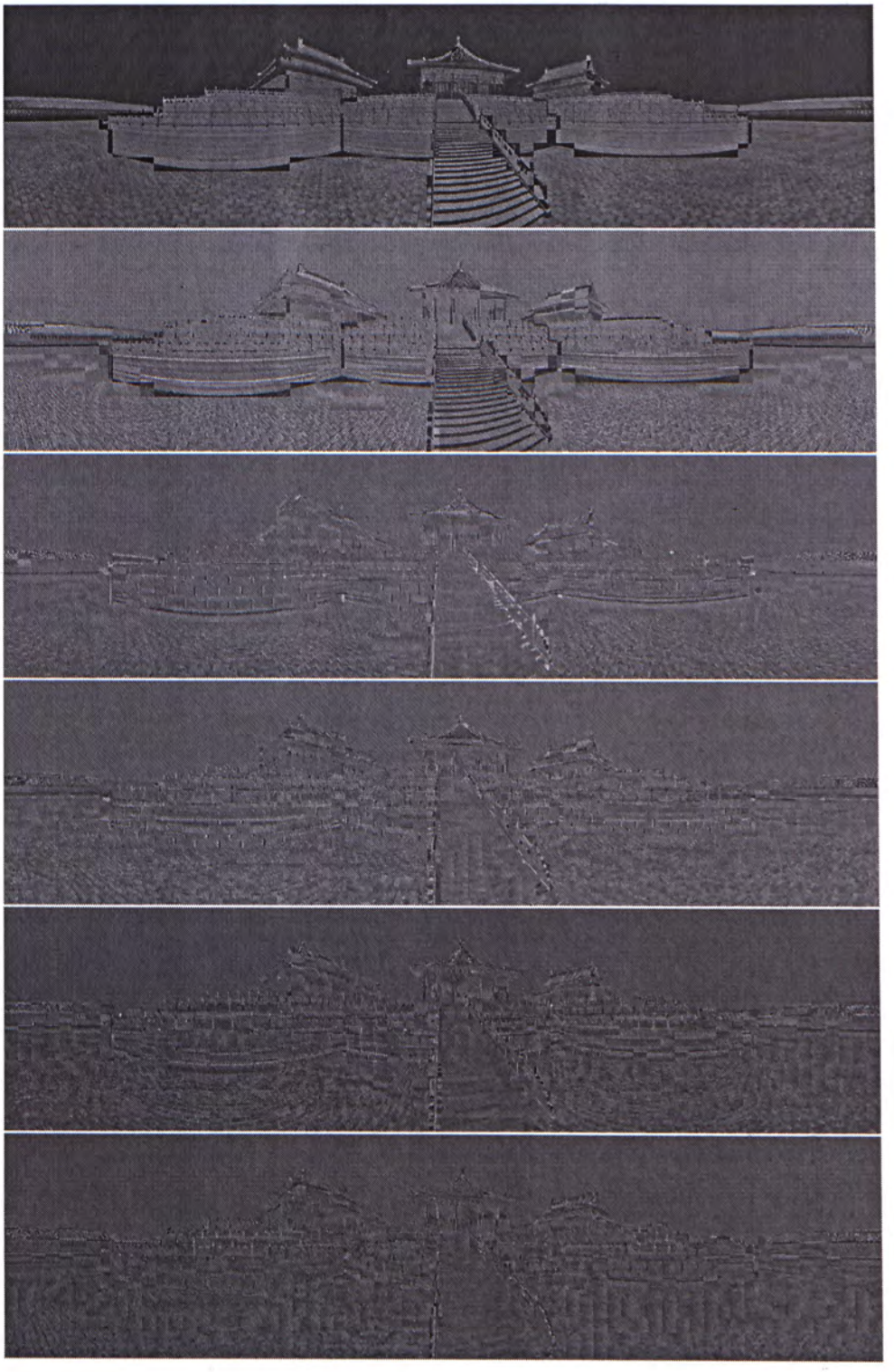


Figure 6.4: The first 6 tiled eigenimages (Y channel) of 'forbidden'.

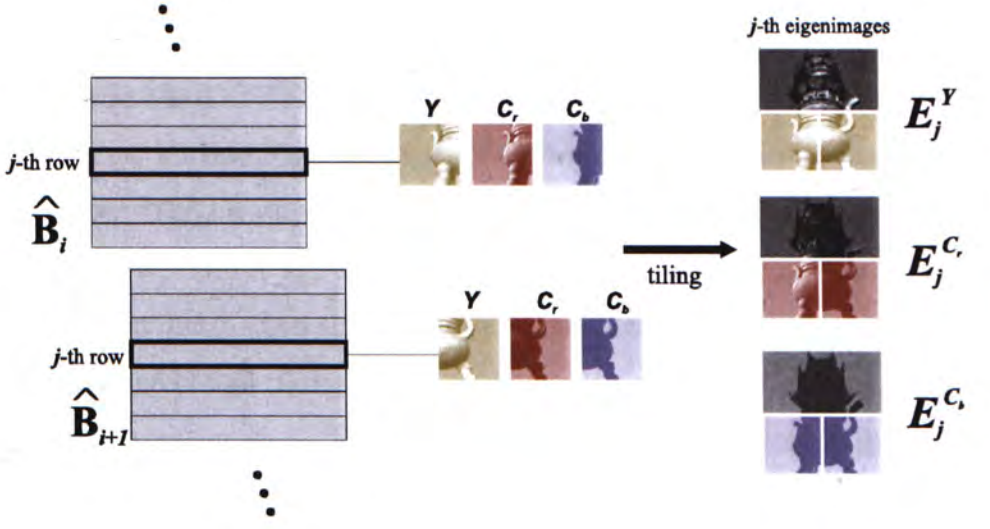


Figure 6.5: Matrix $\hat{\mathbf{B}}$ is rebinned to form the tiled eigenimages.

where $u, v = 0, 1, 2, ..., N-1$, $f(x, y)$ denotes the image block, and $C(u, v)$ denotes the transformed DCT coefficients. The backward transform is

$$f(x,y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v) \cos\frac{(2x+1)u\pi}{2N} \cos\frac{(2y+1)v\pi}{2N} \tag{6.2}$$

where $x, y = 0, 1, 2, ..., N-1$. In both Equations 6.1 and 6.2, α is

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } u = 0 \\ \sqrt{\frac{2}{N}} & \text{if } u = 1, 2, ..., N-1 \end{cases} \tag{6.3}$$

The parameter N is the DCT block size (assuming the block is a square). In our case, since we use DCT block size 8×8, therefore N is equal to 8. Note that this DCT block size is different from the block size mentioned in Chapter 4. Since we apply 8×8 block-based DCT on the tiled eigenimages, boundaries between blocks (16×16) in the eigenimages shall not introduce any artifact.

To DC transform an image (image block) means to transform the values of the image from the spatial domain to the frequency domain. As real-life images are usually consisted of low frequency data, so we can allocate fewer

storage space to high frequency data in order to compress the data. We can allocate fewer storage space to high frequency data and more to low frequency data through quantization.

Bit Allocation and Quantization

To quantize the DCT coefficients, we apply non-uniform quantization. An overall target bit rate R_B is specified for the whole eigenimage encoding process by the user. The bit rate is then sub-allocated to three channels according to the ratio of $E_j^Y : E_j^{C_r} : E_j^{C_b} = 2 : 1 : 1$, since human vision is more sensitive to luminous channel Y than the chrominuous channels C_r and C_b. Same bit rate is then assigned to all tiled eigenimages in the same channel (in despite of their orders). The rationale is that we believe it is visually important to preserve high-frequency details in high-order eigenimages even though they contribute less energy. For example, if $R_B = 3.0$, then bit rates for E_j^Y, $E_j^{C_r}$, and $E_j^{C_b}$ are 4.5, 2.25, and 2.25 for every j respectively. That is,

channel \ eigenimage	1^{st}	2^{nd}	3^{rd}	...
Y	4.5	4.5	4.5	...
C_r	2.25	2.25	2.25	...
C_b	2.25	2.25	2.25	...

For each tiled eigenimage, we use DCT block size 8×8, so 64 data sources (one for each DCT coefficient) are formed. The number of samples in each data source is equal to the number of blocks in a single image. Each source is quantized by the generalized Lloyd algorithm [42]. The bit rate of each source is calculated in the following manner. The distortion for a data source S_μ [43] can be modeled by

$$D(R_\mu) \approx \alpha_\mu \sigma_\mu^2 2^{-2R_\mu},$$

where D is the distortion, R_μ is the encoding rate in bits/sample, σ_μ^2 is the variance of the source, and α_μ is a user-defined constant.

Constant α_μ depends on the density of source as well as the type of encoding method used. In this thesis, we assume that it is independent of the encoding rate. The DC (first) component of DCT is assumed to have a Gaussian distribution as described in [44]. All other sources are modeled as Laplacian sources. Empirically, constant α_μ is chosen to be 2.7 and 4.5 for Gaussian and Laplacian sources respectively [45].

Following the common practice in image compression [43] [45] [46], we let R_B be the target bit rate and H be the number of sources. Let N be total number of values in an eigenimage, N_μ be the number of samples in source S_μ, and $p_\mu = N_\mu/N$ be the normalized weight for source S_μ. We have

$$\begin{aligned} \sum_{\mu=0}^{H-1} N_\mu &= N \\ \sum_{\mu=0}^{H-1} p_\mu &= 1 . \end{aligned}$$

The bit allocation problem can be regarded as a minimization of the following function,

$$\min \sum_{\mu=0}^{H-1} p_\mu D(R_\mu) \tag{6.4}$$

subject to the constraint

$$\sum_{\mu=0}^{H-1} p_\mu R_\mu = R_B .$$

Using the Lagrangian multiplier techniques, the solution of the above optimization problem is

$$R_\mu = R_B + \frac{1}{2}\log_2(\alpha_\mu \sigma_\mu^2) - \{\sum_{\nu=0}^{H-1} \frac{p_\nu}{2}\log_2(\alpha_\nu \sigma_\nu^2)\} . \tag{6.5}$$

Intuitively speaking, the number of bit allocated to a source should be increased if the variance of samples in that source is large. After quantization, we perform arithmetic coding [47] on the quantized values to further reduce the data size. The usual reduction of this entropy coding is around 5% - 6%.

6.1.2 Discrete Wavelet Transform

There is a trend of using discrete wavelet transform in image coding since the introduction of JPEG 2000 standard [48]. DWT, which is used in JPEG 2000, preserves better image quality in low bit rate than DCT, which is used in JPEG, does.

While discrete cosine transform is analyzing the frequency, discrete wavelet transform is analyzing the scale. The generic form of wavelet transform is an 1-D wavelet transform. The overview of it is illustrated in Figure 6.6. An input signal, or vector, is first passed to the low-pass and high-pass filters, which we name them as h and g respectively. The output vectors are then downsampled by a factor of two. For example, an input vector with length 8 will have an output vector with length 4. The output vectors from the low-pass filter are the approximation coefficients while the output vectors from the high-pass filter are the detail coefficients.

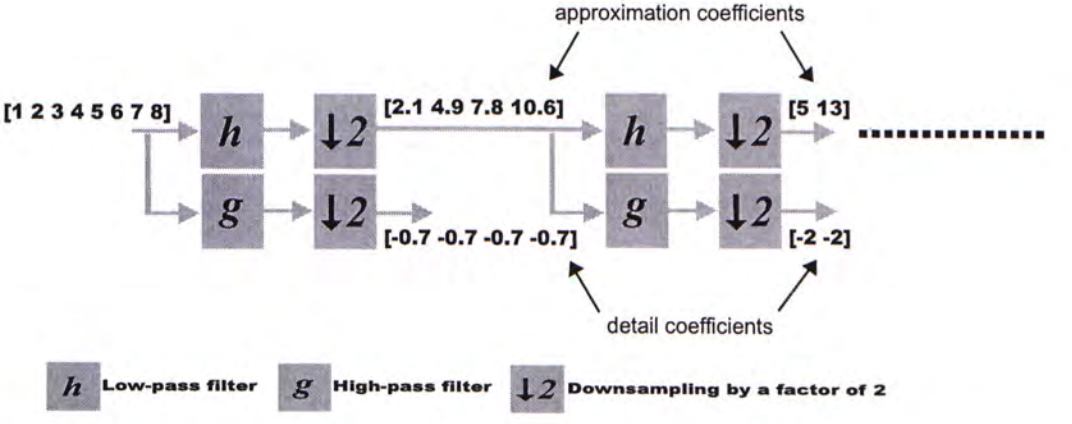


Figure 6.6: The 1-D wavelet decomposition.

By passing through one set of low-pass and high-pass filters, we say that one-level wavelet transform is done. Two-level wavelet transform can be done by passing the approximation coefficients to the low-pass and high-pass filters again. By this, we have a new set of approximation coefficients and detail

coefficients pair. While the old detail coefficients will be kept, the old approximation coefficients are abandoned. In other words, i-level wavelet transform gives 1 set of approximation coefficients and i sets of detail coefficients. Theoretically, we can have infinite levels. However, as the input vectors are finite and the length is discrete, we can have at most $\log_2 l$ levels, where l denotes the input length.

With separable filters, 2-D wavelet transform can be computed by using 1-D transform. Figure 6.7 depicts an overview of an one-level 2-D wavelet decomposition. Each row of the input is first applied an 1-D wavelet transform (through the high-pass and low-pass filters). Each column of the outputs is then applied 1-D wavelet transform vertically. We have totally four sets of outputs, one for the approximation coefficients and three for the detail coefficients: the approximation coefficients (low-pass on row and column), the horizontal detail coefficients (high-pass on row, low-pass on column), the vertical detail coefficients (low-pass on row, high-pass on column), and the diagonal detail coefficients (high-pass on row and column). Two-level 2-D wavelet transform can be done by repeating the process on the approximation coefficients. Figures 6.8 and 6.9 show the results of one-level and two-level 2-D wavelet transforms using Daubechies 9/7 filters on the data sets 'ding' and 'cover' respectively. The outputs are tiled in the following manner

Approximation coefficients	Horizontal detail coefficients
Vertical detail coefficients	Diagonal detail coefficients

The coefficients are then quantized and coded. There are many algorithms for coding the coefficients, for example the embedded zero-tree wavelet coding (EZW) [49]. We have adopted the DWT package "EPWIC" developed by Buccigrossi and Simoncelli [50] in our system.

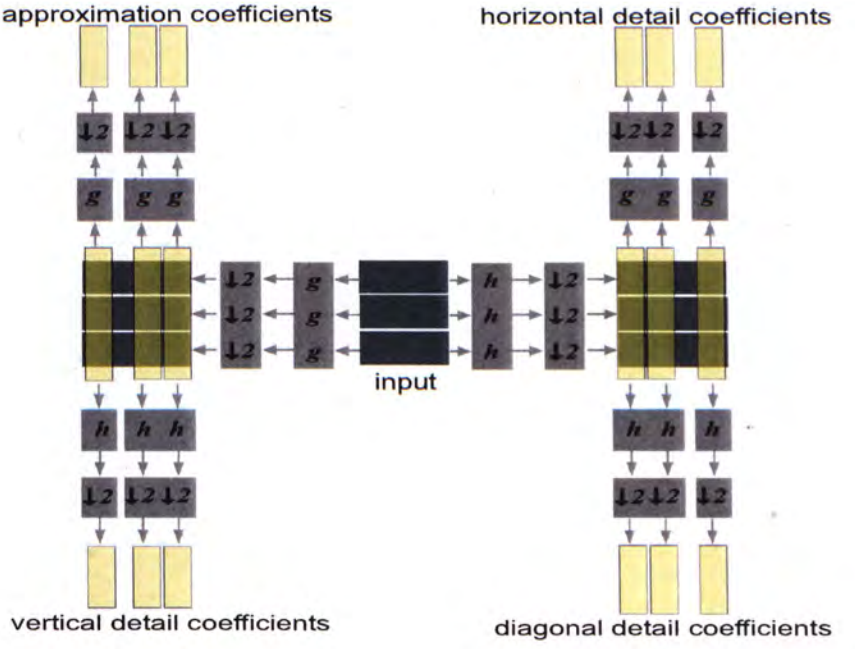
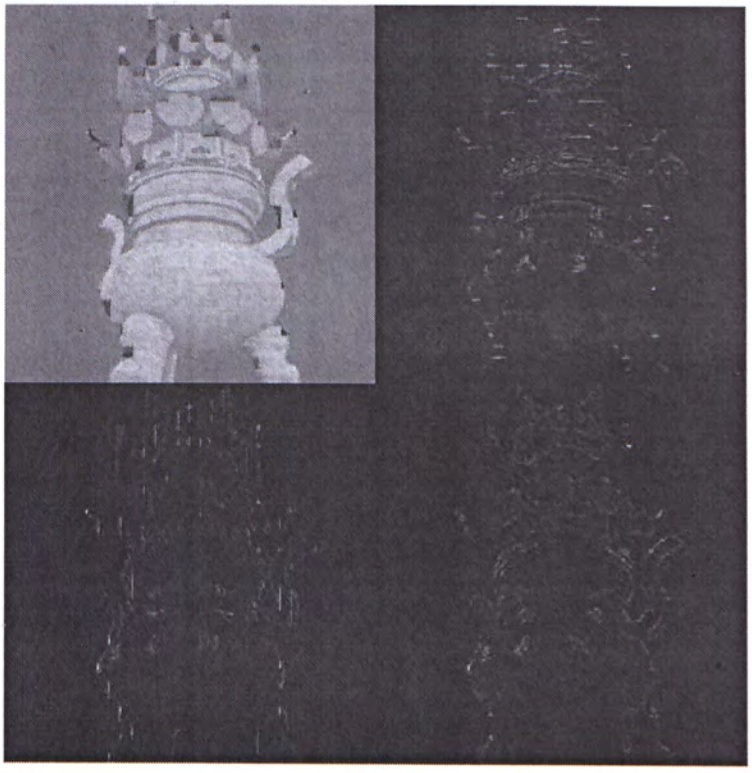


Figure 6.7: One-level 2-D wavelet decomposition.

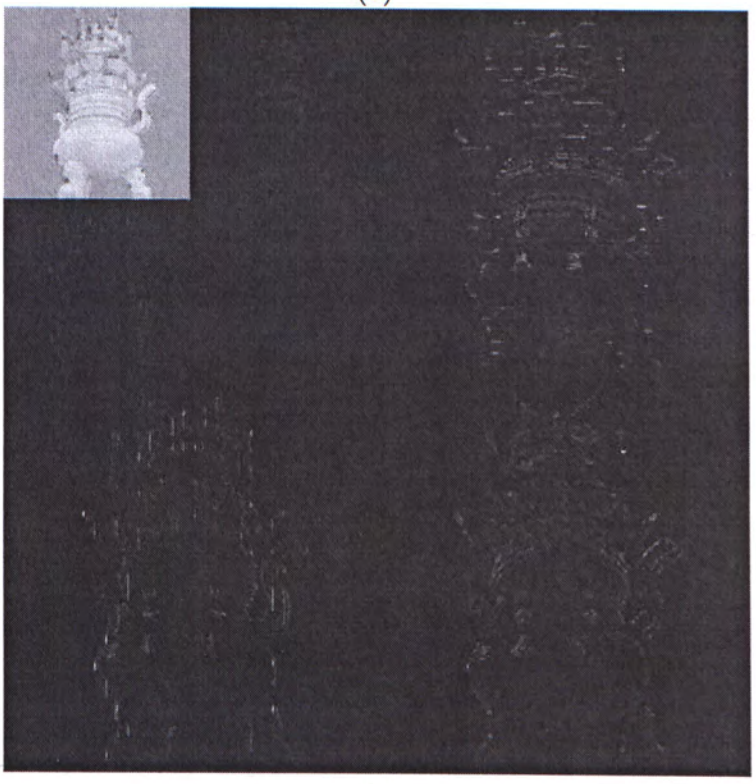
6.2 Evaluation

6.2.1 Statistical Evaluation

To evaluate the performance of our eigenimage coding, we measure the reconstruction error versus the overall target bit rate. The target bit rate we measure only accounts for transform coding and quantization, but excludes the contribution of arithmetic coding. We reconstruct m images using *compressed* eigenimages in $\widehat{\mathbf{B}}$ and *uncompressed* relighting coefficients in $\widehat{\mathbf{A}}$. In the 'ding' example, $m = 1200$. The m reconstructed images are compared with m control images to compute the PSNR. To investigate the error solely due to eigenimage coding, we use the images reconstructed from *uncompressed* $\widehat{\mathbf{A}}$ and *uncompressed* $\widehat{\mathbf{B}}$ as the control images. Figures 6.10 and 6.11 show the statistics of four data sets. From the figures, we can find that DWT outperforms DCT especially in low bit rates. When bit rate is around 1, DWT



(a)



(b)

Figure 6.8: (a) One-level and (b) two-level 2-D wavelet transforms using Daubechies 9/7 wavelet for 'ding'.

(a)

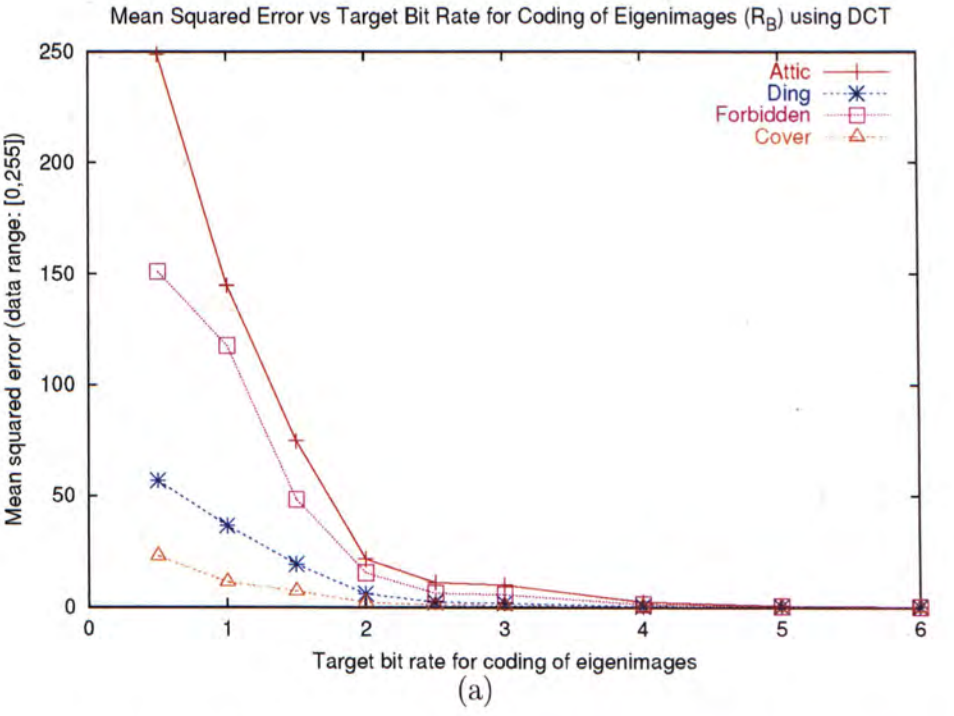
(b)

Figure 6.9: (a) One-level and (b) two-level 2-D wavelet transforms using Daubechies 9/7 wavelet for 'cover'.

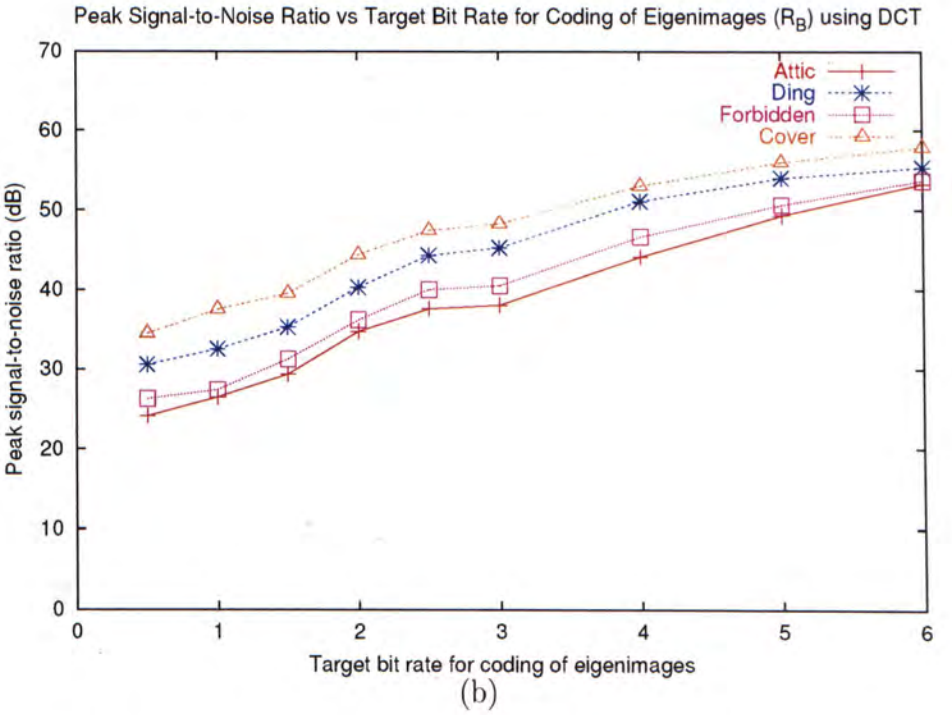
can reach about 40dB (mean squared error below 25) already while DCT can reach about 30dB only. Among the four data sets, performance on data sets 'attic' is the worst in both DWT and DCT cases. It is because the scenery of 'attic' is the most complex one among the four data sets. On the other hand, performance on data sets 'cover' is the best one because there is large region of dark area and the scenery is relatively simpler. As DWT performs better than DCT, especially in low bit rate, we suggest to use DWT for compressing the eigenimages in this phase.

6.2.2 Visual Evaluation

Besides the statistical evaluation, we also evaluate the coding visually. Figure 6.12 compares the reconstructed images from the forbidden city data set using DCT in a side-by-side manner. From left to right, top to bottom, the target bit rate increases from 0.1 to 4.0. When a low bit rate is specified, the reconstruction exhibits blocky artifact and image details are lost. Since we apply DCT-based coding to eigenimages, the visual artifact is similar to the artifact of JPEG images at low bit rate. The quality improves as the bit rate increases. There is no significant artifact when the bit rate is raised to 3.0. Figure 6.13 compares reconstructed images from the same data set but using DWT this time. The target bit rate increases from 0.1 to 1.0. By using DWT, a relatively lower bit rate can achieve a better reconstruction quality. It does not have the blocky artifacts occurred in DCT coding. This confirms to the founding of using DWT for low bit rate coding of images. From the above figure and statistics, 3.0 bit is a cost-effective choice for eigenimage coding using DCT as it returns more than 35 dB in each test case. However, 1.0 bit is enough if DWT is used in eigenimage coding.

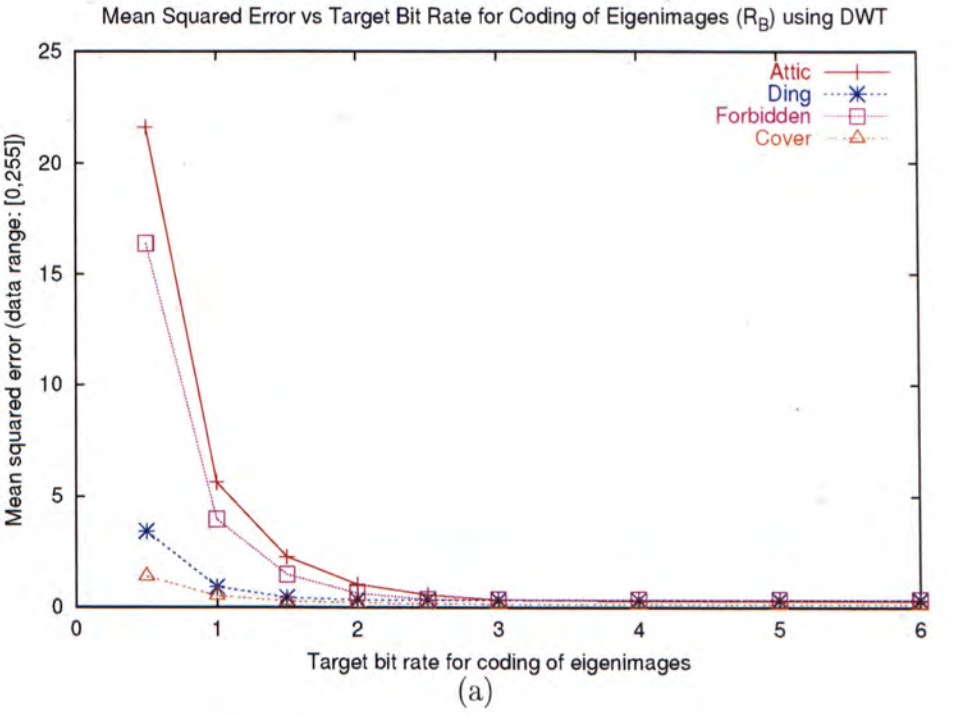


(a)

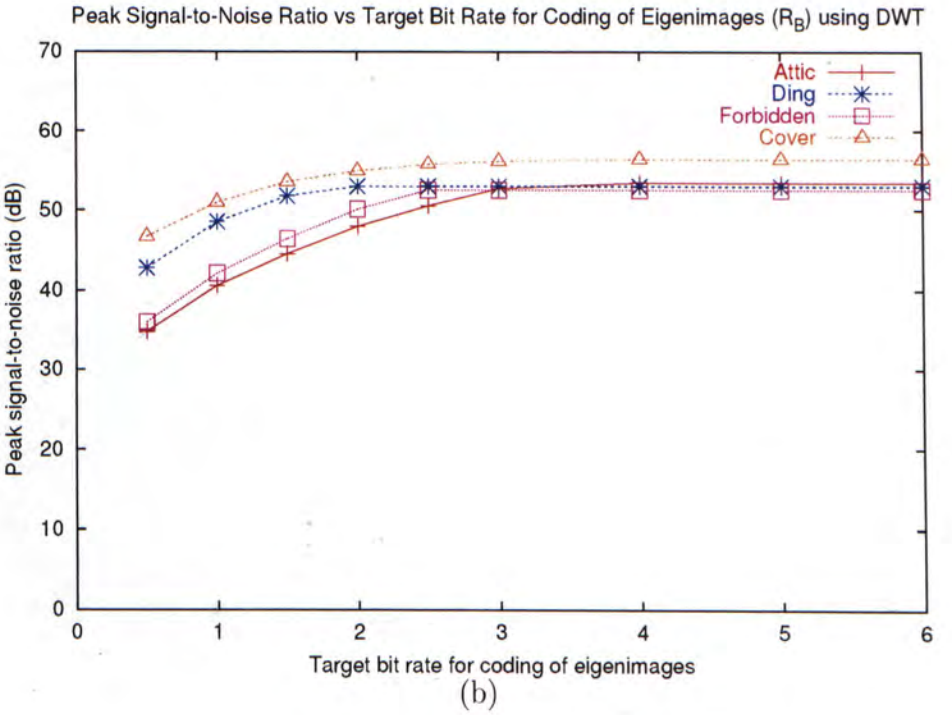


(b)

Figure 6.10: PSNR versus target bit rate for eigenimage coding (R_B) using DCT.



(a)



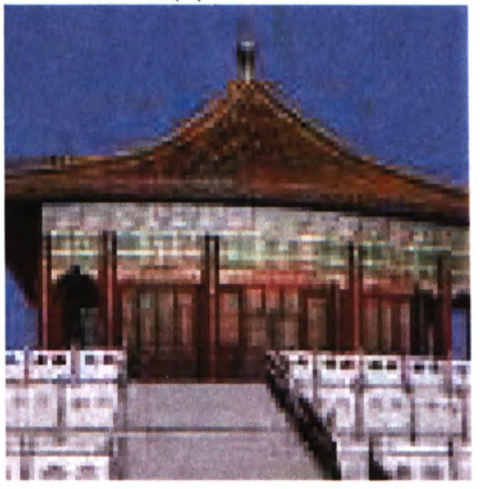
(b)

Figure 6.11: PSNR versus target bit rate for eigenimage coding (R_B) using DWT.

(a) 0.1 bit

(b) 1.0 bit

(c) 2.0 bit



(d) 3.0 bit

(e) 4.0 bit

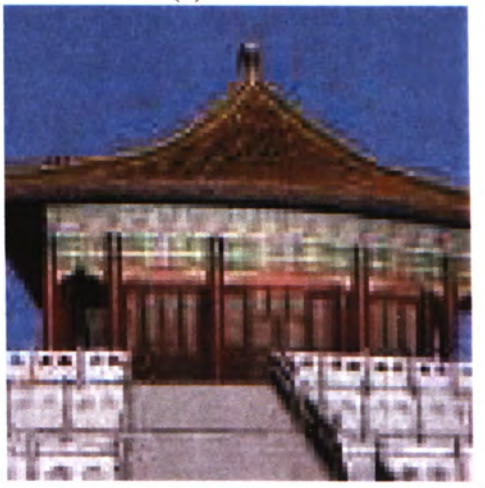
Figure 6.12: Visual artifacts introduced by encoding eigenimages using DCT with different target bit rates.

(a) 0.1 bit

(b) 0.3 bit

(c) 0.5 bit

(d) 0.7 bit



(e) 1.0 bit

Figure 6.13: Visual artifacts introduced by encoding eigenimages using DWT with different target bit rates.

Chapter 7

Relighting Coefficient Coding

After discussed how we compress eigenimages, we will discuss how we compress the relighting coefficients obtained after PCA in this chapter. As relighting coefficients are not like eigenimages in nature, we should not compress them using the same method. Uniform quantization is suggested for compressing them. Again, we will give an evaluation on the performance of compressing relighting coefficients using the suggested method at the end of this chapter.

7.1 Quantization and Bit Allocation

Unlike the eigenimage, the relighting coefficients in $\widehat{\mathbf{A}}_i$ for $i = 0, \cdots, w-1$ do not resemble an image. The total number of coefficients is much larger than the total number of pixels in eigenimages. It may not be practical to compress them using non-uniform quantization as the training of codebook is time-consuming. Instead, we encode them using uniform quantization.

We find that the variances of values from the same column of different $\widehat{\mathbf{A}}_i$ for $i = 0, \cdots, w-1$ are similar. Hence we group them into the same data source. Figure 7.1 illustrates how data sources are formed. Since there are k columns in $\widehat{\mathbf{A}}_i$, there are k data sources. The user specifies an overall target bit rate, R_A, for encoding the relighting coefficients.

Again we would like to minimize the total distortion error in Equation 6.4 [43].

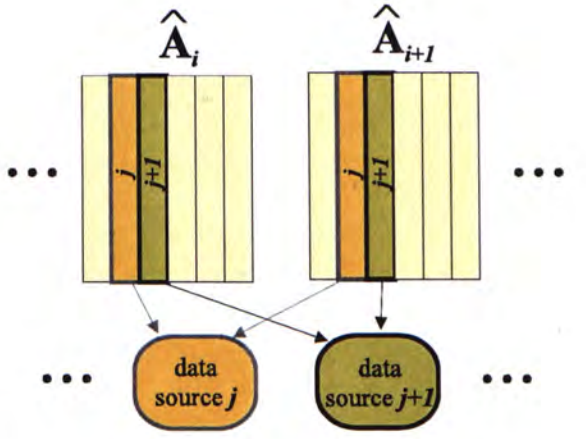


Figure 7.1: Each data source is formed by grouping elements from the same columns in different $\widehat{\mathbf{A}}_i$.

Just like the bit allocation in Section 6.1.1, the optimal bit allocation for each source S_μ is,

$$R_\mu = R_A + \frac{1}{2}\log_2(\alpha_\mu \sigma_\mu^2) - \{\sum_{\nu=0}^{k-1} \frac{1}{2k}\log_2(\alpha_\nu \sigma_\nu^2)\}\,. \tag{7.1}$$

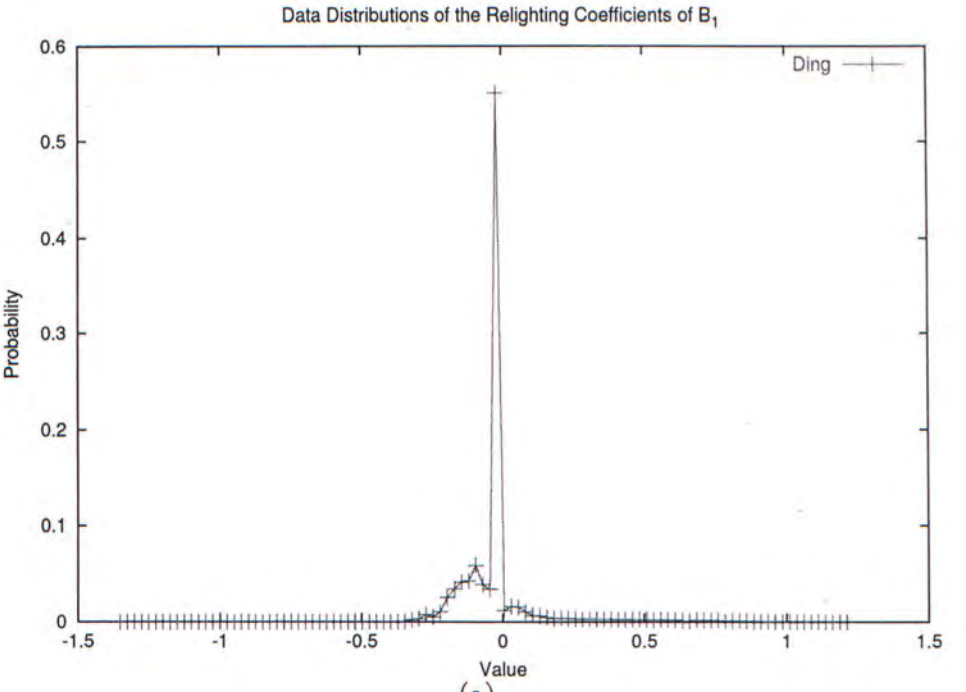
The above equation is different from Equation 6.5 as R_B is now replaced by R_A. Moreover, as all k data sources contain same number of coefficients, we can substitute $H = k$ and $p_\mu = 1/k$. Figure 7.2 shows the histograms of the first 6 data sources from data set 'ding'. All of them exhibit a Laplacian distribution. Hence we set constant α_μ to be 4.5.

Then we can determine the step size $\triangle_\mu$ for the μ-th source by

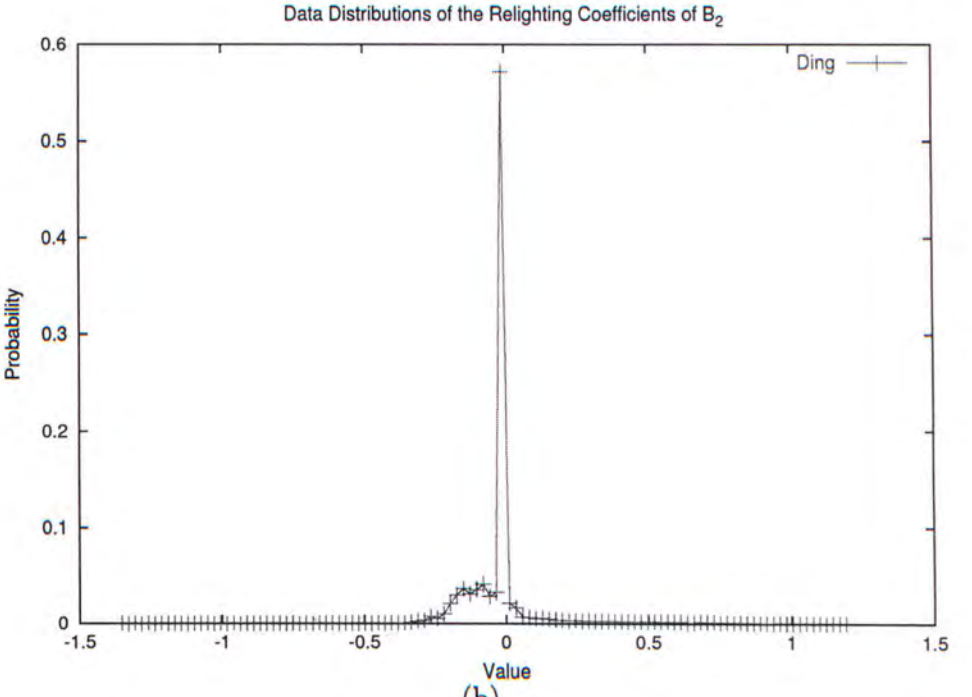
$$\triangle_\mu^2 = \frac{|\max_\mu|^2}{2^{R_\mu}} \tag{7.2}$$

where $\max_\mu$ is the maximum value of the μ-th source.

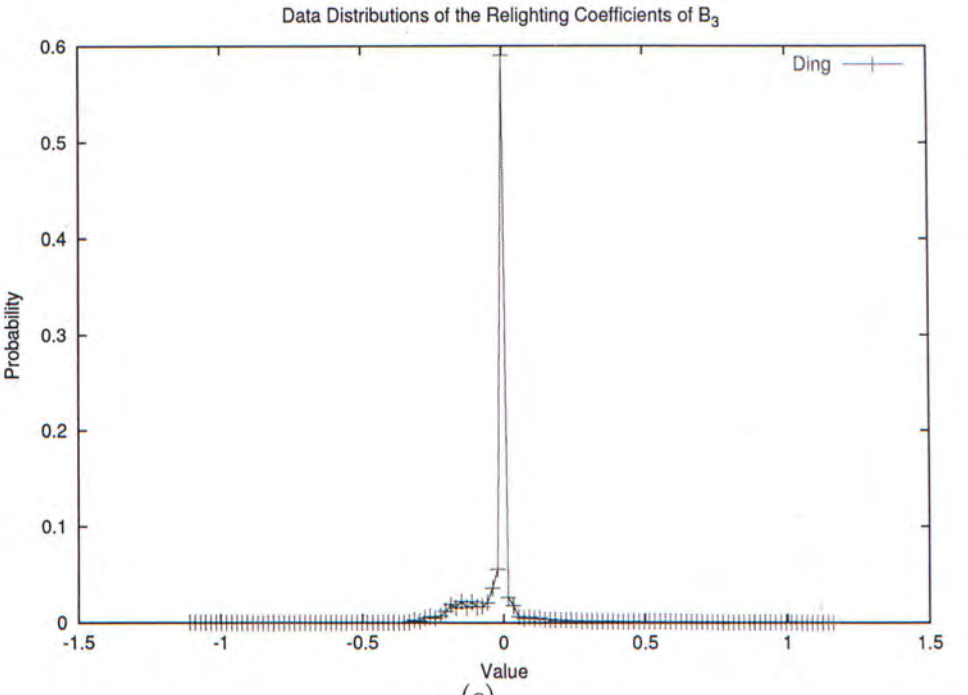
Just like the eigenimage encoding, the quantized relighting coefficients are further coded using arithmetic coding to reduce the storage requirement. The rough reduction due to arithmetic coding is around 35%.


Data Distributions of the Relighting Coefficients of B_1
Ding
Probability
Value
0.6
0.5
0.4
0.3
0.2
0.1
0
-1.5
-1
-0.5
0
0.5
1
1.5

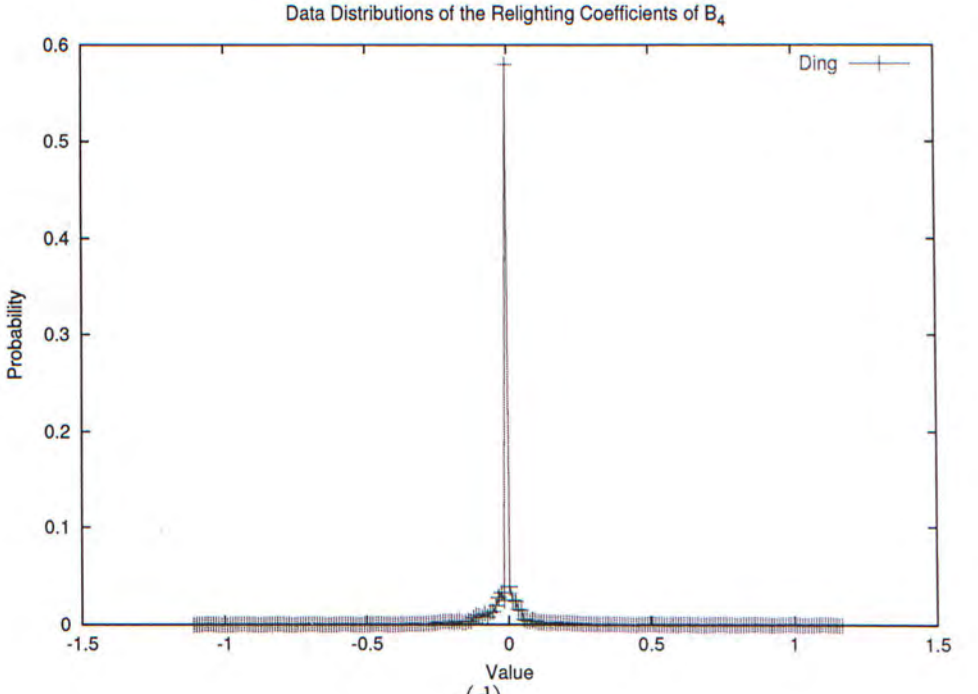
(a)


Data Distributions of the Relighting Coefficients of B_2
Ding
Probability
Value
0.6
0.5
0.4
0.3
0.2
0.1
0
-1.5
-1
-0.5
0
0.5
1
1.5

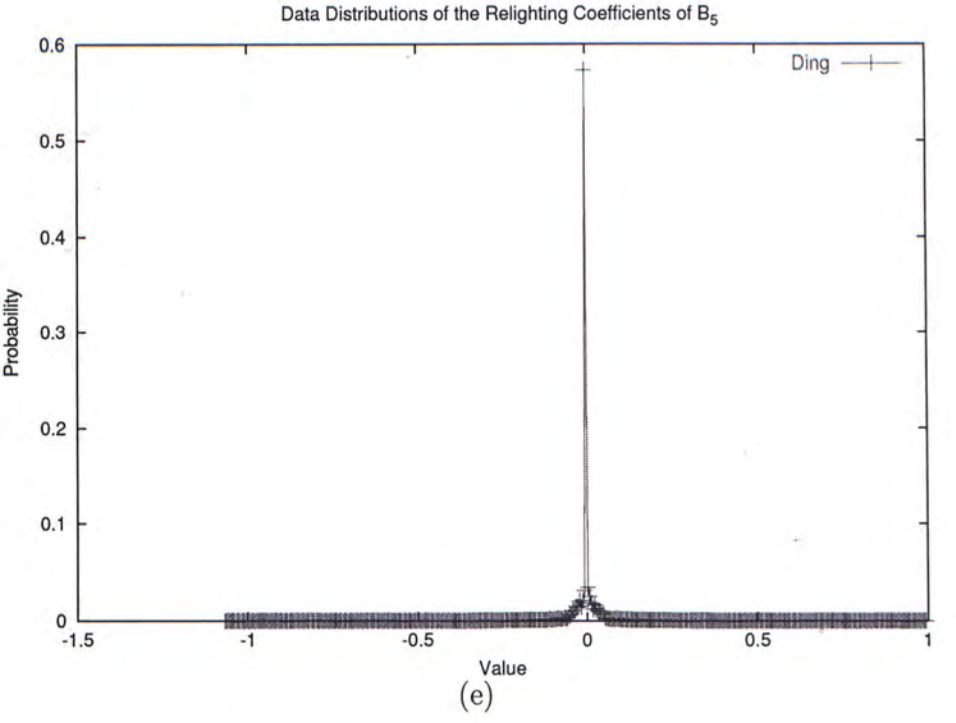
(b)


Data Distributions of the Relighting Coefficients of B_3
Ding
Probability
Value
0.6
0.5
0.4
0.3
0.2
0.1
0
-1.5
-1
-0.5
0
0.5
1
1.5

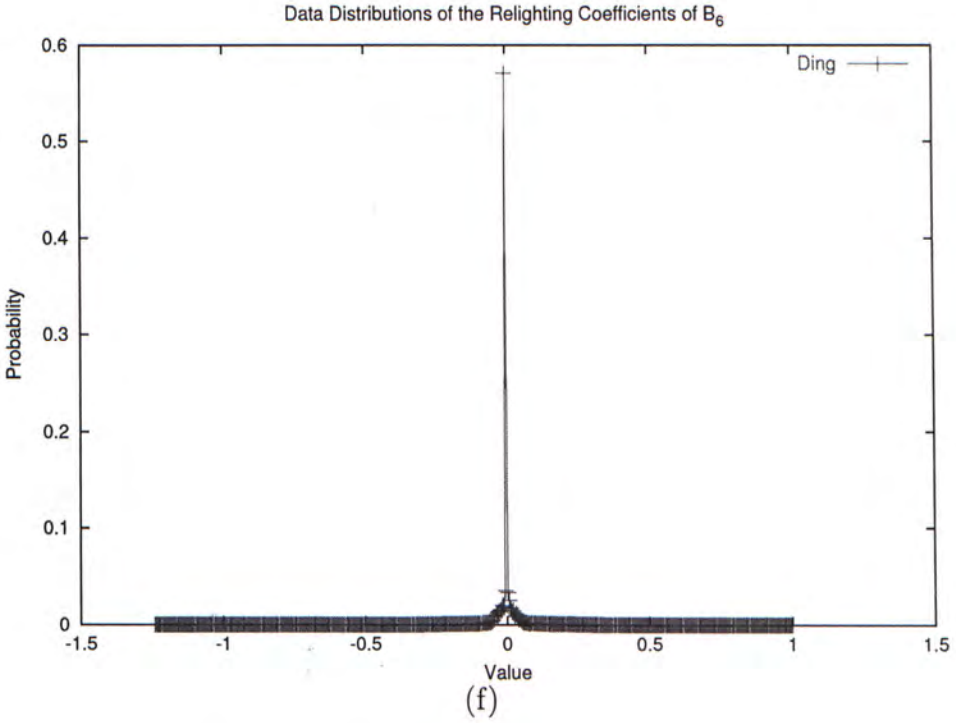
(c)


Data Distributions of the Relighting Coefficients of B_4
Ding
Probability
Value
0.6
0.5
0.4
0.3
0.2
0.1
0
-1.5
-1
-0.5
0
0.5
1
1.5

(d)



(e)



(f)

Figure 7.2: Histograms of the first six data sources.

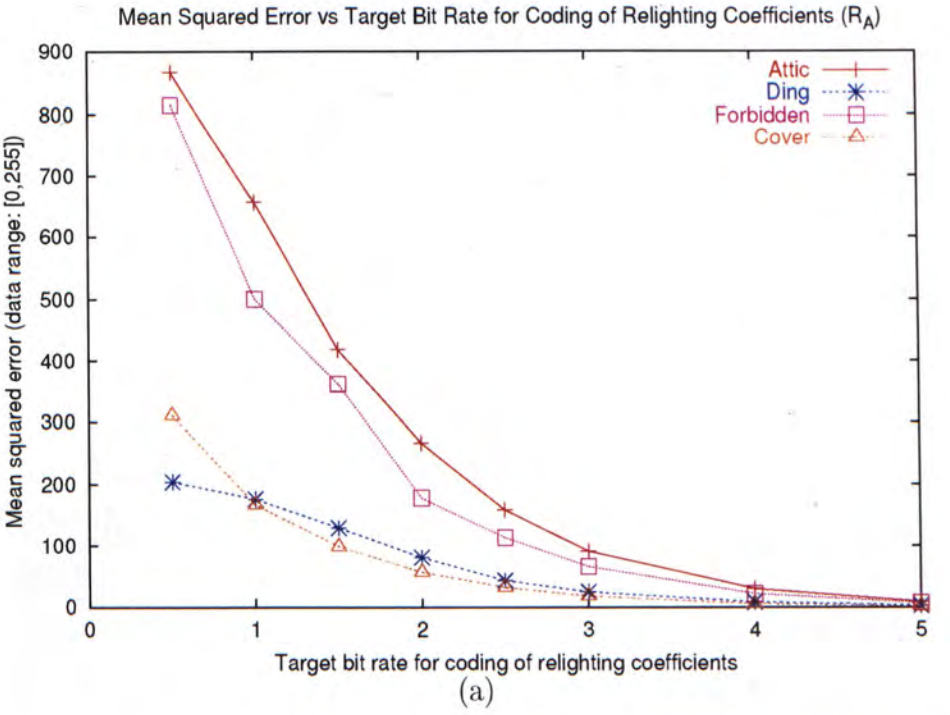
7.2 Evaluation

7.2.1 Statistical Evaluation

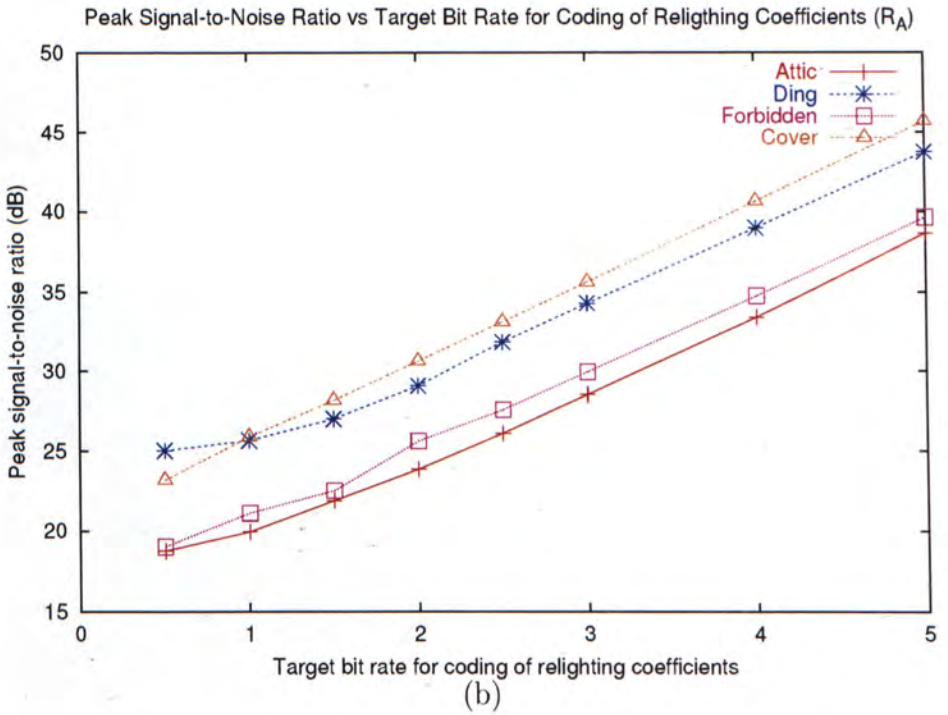
We measure the reconstruction error solely due to the encoding of relighting coefficients. To do so, we reconstruct m images from *compressed* $\widehat{\mathbf{A}}$ and *uncompressed* $\widehat{\mathbf{B}}$. The reconstructed images are compared to the same control images in Section 6.2. We measure the PSNR as R_A increases (Figure 7.3). From the statistics, we find that the performance in low bit rates is not good as it can only give us a PSNR under 30dB. However, the performance seems improving linearly as shown in Figure 7.3. The slopes of the four tested data sets are almost the same. Just like the results of coding eigenimages, the performance on coding the data set 'attic' is the worst while that on coding the data set 'cover' is the best one. Therefore, the complexity of the scenery seems affecting the performance on coding both eigenimages and relighting coefficients.

7.2.2 Visual Evaluation

Again we evaluate the coding visually. Figure 7.4 shows the reconstructed images from the data set of panoramic forbidden city. Unlike the visual artifact in eigenimage encoding, image details are not lost (the details within each block can still be observed clearly). Instead, the major visual artifact introduced at low bit rate is the tone inconsistency among the neighboring blocks. This is because relighting coefficients are mainly responsible for information in the illumination dimension. Low bit rate in coding relighting coefficients in some sense is analogous to the undersampling of illumination dimension. The tone inconsistency becomes unobservable when the target bit rate is increased to 4.0. From the above figure and statistics, 4.0 bit is a good choice among the tested data sets as it gives more than 30dB and reduces the tone inconsistency.

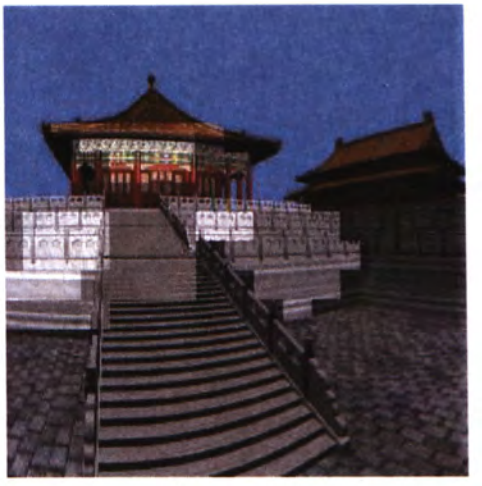


(a)

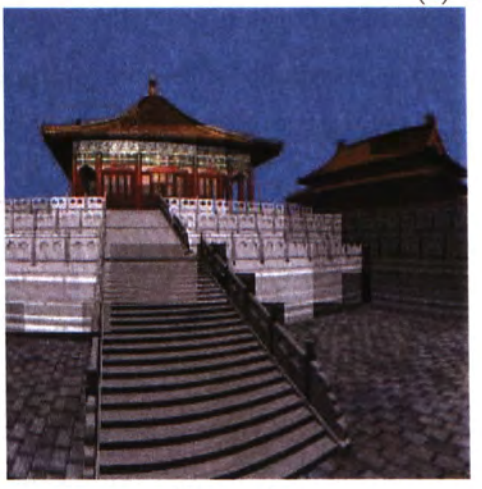


(b)

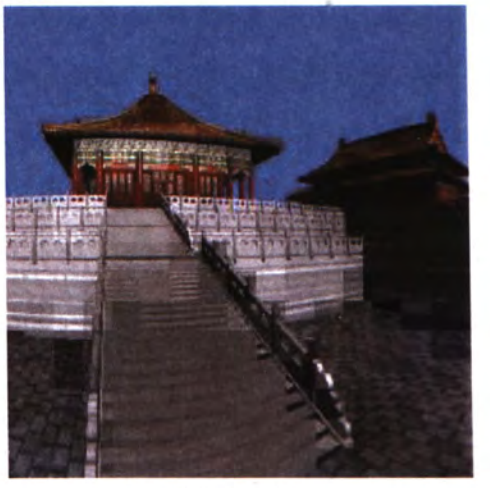
Figure 7.3: PSNR versus target bit rate for coding of relighting coefficients (R_A).



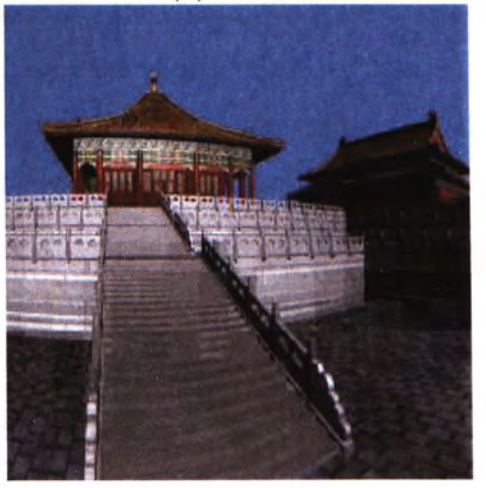
(a) 0.1 bit



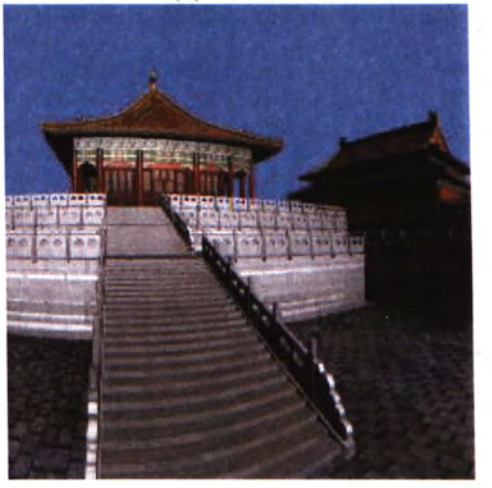
(b) 1.0 bit



(c) 2.0 bit



(b) 3.0 bit



(c) 4.0 bit

Figure 7.4: Visual artifacts introduced by encoding relighting coefficients with different target bit rates.

Chapter 8

Relighting

After discussing how we compress illumination-adjustable images in the previous chapters, we are going to discuss in this chapter how we can relight images given the illumination-adjustable images compressed using our method, or we can say, using our image-based relighting representation model. Relighting is a decoding procedure. However, if we *decompress* the compressed illumination-adjustable images, we mean to reconstruct all the input data. On the other hand, if we *relight* an image from the compressed illumination-adjustable images, we try to reconstruct the image at a desired lighting condition only. This requires the random accessibility of our representation model. The latter scenario is more meaningful and useful as we are often interested in one particular lighting condition at a moment only. Relighting involves the processes of interpolation and reconstruction. In designing the relighting engine, our primary concern is the speed. We trade the memory consumption for speed. To do so, we keep the eigenimages and the set of relighting coefficients in memory for fast reconstruction. For the rest of this chapter, we will discuss the relighting procedure in details. The relighting procedure is divided into two phases, namely first-phase and second-phase decoding. For second-phase decoding, we have two implementations, software relighting and hardware-assisted relighting. Hardware-assisted relighting helps us in achieving real-time relighting by using commodity hardware display boards.

8.1 Overview

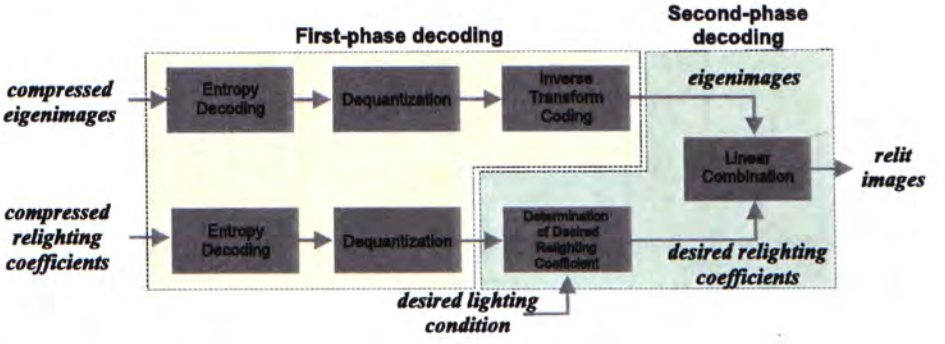


Figure 8.1: Relighting procedure.

The relighting procedure is revealed in Figure 8.1 (same as Figure 3.5(b)). It can be divided into two major phases. In the first phase, the compressed eigenimages and relighting coefficients are decompressed into memory. Since the image is usually relit under a user-defined lighting condition, we only need to reconstruct a limited number of images each time. Hence in the second phase, the relighting engine interpolates the relighting coefficients and synthesizes the desired image by linear combination.

8.2 First-Phase Decoding

All preprocesses are done in the first-phase decoding. In this phase, the relighting engine expands all compressed eigenimages and the relighting coefficients into main memory (Figure 8.1). Note that the decoding is done only once during the initialization. This allows the second-phase decoding to access the necessary eigenimages and coefficients in real-time.

Recall that the block-wise eigenimages are rebinned to form an image-wise eigenimages before the transform coding. Hence there is no need to rebin the decompressed eigenimages as they are already in an image form. The

only process needed is to convert them from YC_rC_b to RGB color space. As the reconstruction is a linear process (linear combination of eigenimages), the color transform can be simply applied to the eigenimages instead of the reconstructed images. The correctness of this color transform is not difficult to show by the following derivation.

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix} \begin{bmatrix} Y \\ C_r \\ C_b \end{bmatrix}$$

where (R, G, B) and (Y, C_r, C_b) are the reconstructed pixel values in RGB and YC_rC_b spaces respectively, c_{ij}'s are elements in the color transformation matrix. Each reconstructed pixel value is the linear combination of, b_j^i's, pixel values from the eigenimages, *i.e.*

$$\begin{aligned} Y &= a_0 b_0^Y + a_1 b_1^Y + \cdots + a_{k-1} b_{k-1}^Y \\ C_r &= a_0 b_0^{C_r} + a_1 b_1^{C_r} + \cdots + a_{k-1} b_{k-1}^{C_r} \\ C_b &= a_0 b_0^{C_b} + a_1 b_1^{C_b} + \cdots + a_{k-1} b_{k-1}^{C_b} \end{aligned}$$

where a_j is the relighting coefficient from matrix $\mathbf{A}$ and b_j^i is the i-channel pixel value from the j-th eigenimage in matrix $\mathbf{B}$. By reordering the terms, we can rewrite R as

$$\begin{aligned} R &= c_{00}Y + c_{01}C_r + c_{02}C_b \\ &= a_0(c_{00}b_0^Y + c_{01}b_0^{C_r} + c_{02}b_0^{C_b}) + \\ &\quad a_1(c_{00}b_1^Y + c_{01}b_1^{C_r} + c_{02}b_1^{C_b}) + \\ &\quad \cdots \\ &\quad a_{k-1}(c_{00}b_{k-1}^Y + c_{01}b_{k-1}^{C_r} + c_{02}b_{k-1}^{C_b}) \\ &= a_0 b_0^R + a_1 b_1^R + \cdots + a_{k-1} b_{k-1}^R \end{aligned}$$

where b_j^R is the pixel value from the j-th eigenimage transformed to RGB color space. Similar derivation can be applied to channels G and B. As the

number of eigenimages is small (in our tested cases, 9), it is more efficient to color-transform the eigenimages than the reconstructed images. This reduces the workload during relighting.

8.3 Second-Phase Decoding

Second-phase decoding mainly deals with the user request during run-time. Figure 8.2 depicts the reconstruction and interpolation process. For each light vector (θ_l,ϕ_l) specified by the user, at most four neighboring images (blue dots) on the grid points are reconstructed. Each of them is reconstructed by linearly combining the eigenimages with the corresponding relighting coefficients as weights (see Section 5.2 and Figure 5.2). Then these reconstructed images are bi-linearly interpolated to synthesize the desired image (red dot). In practice, there is no need to reconstruct these four images. As the interpolation is linear, we can simply bi-linearly interpolate the four sets of relighting coefficients and synthesize the desired image by linearly combining the eigenimages with the interpolated coefficients as weights. The mean image is then added to shift the values back to the original range (Section 4.3). If there are more than one directional light sources used for relighting, the above process is repeated for each light source and the result is accumulated (summed) to generate the final desired image. Sections 8.3.1 and 8.3.2 describe the procedure in details for software relighting and hardware-assisted relighting respectively.

8.3.1 Software Relighting

Software relighting is only a contrast wording to *hardware-assisted* relighting. It means that we do not need extra hardware assistance in the relighting procedure. However, we are still able to make optimization in order to accelerate the relighting process, although the speed is unable to compare with hardware-assisted relighting.

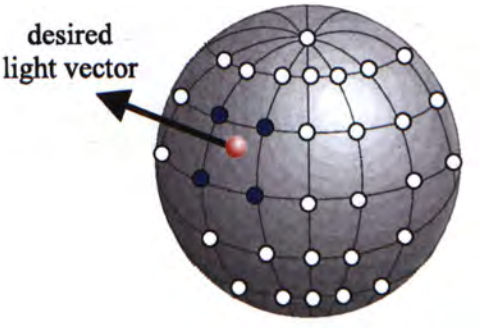


Figure 8.2: The relighting due to one light source (red dot) requires the reconstruction and interpolation of at most four neighboring samples (blue dots).

In order to efficiently relight the image, we arrange the relighting coefficients in the memory so that coefficients being accessed in the same relighting pass are grouped together. When interpolating the relighting coefficients, w relighting coefficients (one for each block, there are w blocks in an image) of the same order are accessed simultaneously. Hence, we group those w coefficients together in order to speed up the memory access. Figure 8.3 shows the arrangement of relighting coefficients for the data set 'ding'. They are the relighting coefficients corresponding to the first eigenimage. It is a matrix with sub-matrices of w coefficients. Since there are at most four sets of relighting coefficients being looked up in one relighting pass, four neighboring sub-matrices (highlighted as boxes in Figure 8.3) may be accessed at the same time.

Interpolation is a must if we want to relight images with lighting directions that do not appear in the samples. To relight an image with lighting direction at (t, p), we need four images with the nearest lighting directions to (t, p), that is the lighting directions at (x, y), $(x, y+1)$, $(x+1, y)$, and $(x+1, y+1)$ in Figure 8.4. As we sample our lighting directions on a regular grid, x is equal to $\lfloor t \rfloor$ while y is equal to $\lfloor p \rfloor$. We use bi-linear interpolation in our system. Let the image with lighting directions at (x, y) be $Im(x, y)$, bi-linear interpolation

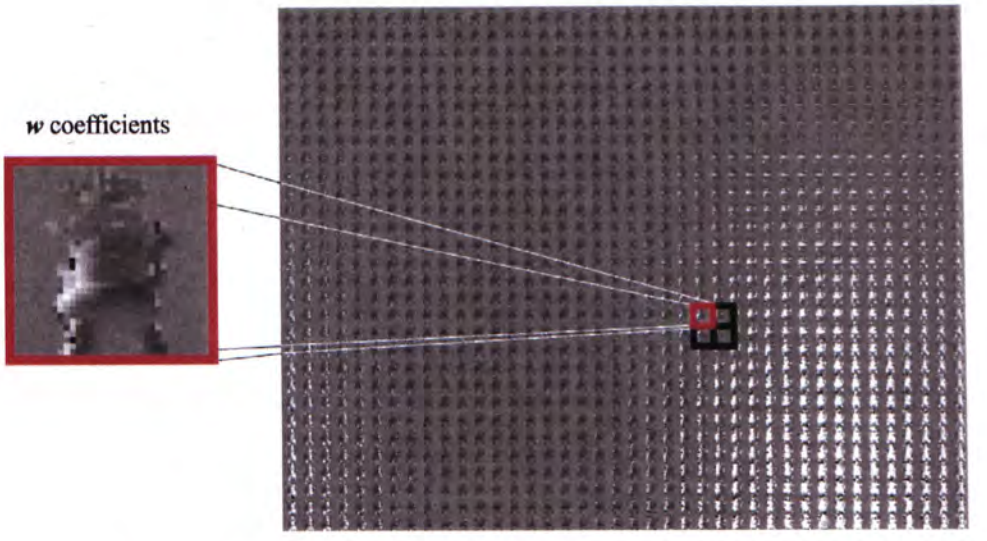


Figure 8.3: Coefficient arrangements for software relighting.

is defined as

$$\begin{aligned} Im(t,p) = \; & (y+1-p)(x+1-t)Im(x,y) + \\ & (y+1-p)(t-x)Im(x+1,y) + \\ & (p-y)(x+1-t)Im(x,y+1) + \\ & (p-y)(t-x)Im(x+1,y+1) \end{aligned} \tag{8.1}$$

In other words, the image $Im(t,p)$ is a linear combination of the images $Im(x,y)$, $Im(x+1,y)$, $Im(x,y+1)$, and $Im(x+1,y+1)$. In practical, we do not perform the bi-linear interpolation in the image domain. Instead, we can bi-linearly interpolate the relighting coefficients thanks to the linear combination property. Just like the reasoning of transforming the eigenimages from YC_rC_B to RGB instead of transforming the relit image (Section 8.2), the correctness of bi-linearly interpolating the relighting coefficients instead of the relit images can be derived easily. We do not include it in this thesis.

Although we can make the above optimizations in the relighting process, the relighting frame rate is still not satisfactory. Using a PIII 800MHz computer, it requires 1 to 2 seconds for relighting an image given a desired lighting

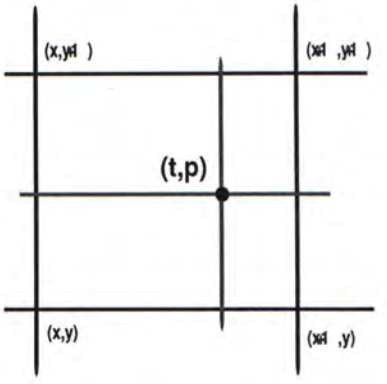


Figure 8.4: Bilinear interpolation.

condition. In next section, we will talk about how we can achieve real-time relighting by using commodity display boards.

8.3.2 Hardware-Assisted Relighting

With the advances of current graphics hardware, we are able to perform parallel operations on consumer-level graphics boards, such as nVidia GeForce 4 or ATI Radeon 8500. These graphics boards are equipped with graphics processing unit (GPU) to execute user-defined *shader* (machine-code like program) [51]. The architecture is basically a SIMD design, a single shader is executed on multiple data (such as pixels or vertices).

This motivates us to utilize the commodity hardware for relighting purpose. Since the major process during relighting is the linear combination of eigen-images, this can be straightforwardly implemented by storing each eigenimage in the texture unit of the hardware. The SIMD-based hardware is efficient in performing parallel per-pixel operations such as multiplication, addition, and subtraction. Pixels in the texture can be manipulated (multiplied, added or subtracted) in parallel. Due to the limitation of our specific hardware, only four texture units can be used in one shader pass. The whole reconstruction requires multiple passes. We are going to describe how we can map our problem

such that we can make use of these display boards.

The Programmable Pixel Shader

The display board we use (nVidia GeForce 3) contains two major units, namely *vertex shader* and *pixel shader*. The vertex shader is responsible for vertex-oriented process. User can define/program their desired actions on the vertices. However, we do not use this shader in our system. Therefore we are not going to introduce anything related to it. On the other hand, we depend heavily on the pixel shader. The pixel shader contains two modules, namely the *texture shader* and the *register combiner*.

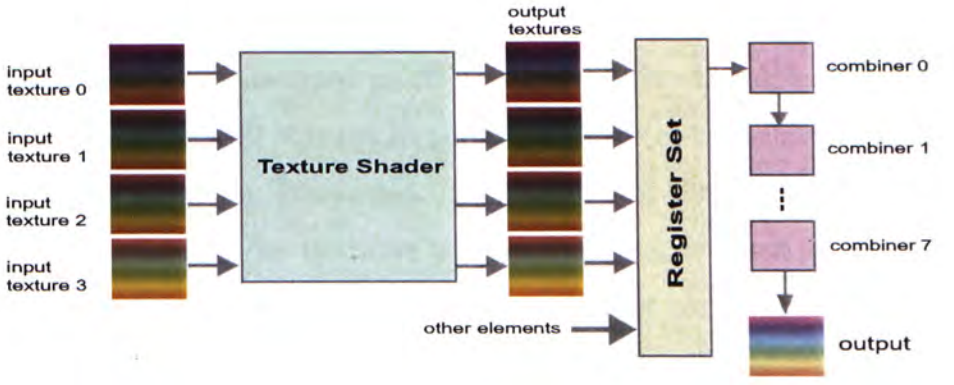


Figure 8.5: The pixel shader.

Texture shader is responsible for texture fetching and filtering. In other words, given a set of input textures (2D arrays of pixels), how we shall look up pixels among them. The look-up process can be done using texture shader operations. Some tricky texture shader operations are used in our implementation, which will be discussed later on. The looked up textures are stored in the output textures. For nVidia GeForce 3, there are totally four input textures and four output textures in the texture shader. Each pixel in an input texture is accompanied by a texture coordinate, which can be defined by users. The output textures are the inputs of the register combiner. Besides, other

elements, such as diffuse color, fog color, and so on, are fed to the register combiner too. However, they are not used in the implementation of our relighting engine. Register combiner is responsible for blending (applying arithmetics on) the fetched or filtered textures. At most eight stages of register combiner actions, which are carried out in series, can be defined. We name one run of texture shader and register combiner as one *shader pass*.

There are several categories of texture shader operations. We will only introduce those operations which are related to our system. One of the operations is conventional textures operation. Actually, it is a definition rather than an operation, although 'operation' is the official naming. The conventional textures can be 1D, 2D, rectangular, or cube map. In our case, 2D and rectangular textures are useful. As their names tell, 2D and rectangular textures can be considered as 2D arrays of pixels. The difference is that 2D textures must be of dimensions which are power of two while rectangular textures can be of any dimensions. Each texture of this category consumes one texture unit. So, we can have at most four such textures in one shader pass. Eigenimages, relighting coefficients, and offset texture are 'defined' as 2D/rectangular textures in our relighting engine. We will introduce the offset texture later on.

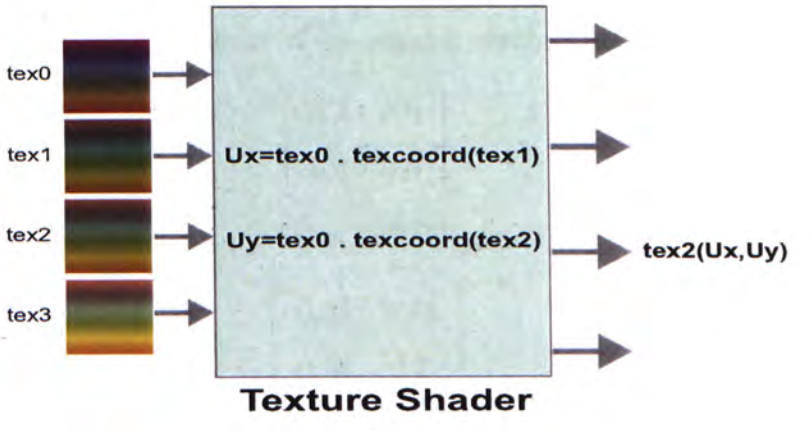


Figure 8.6: 2D dot product look-up operation.

Another useful operation is the dot product look-up operation, which is a bit complicated. It is mainly used for look-up purpose (do not mix up with per-pixel operations). Figure 8.6 illustrates the 2D dot product look-up operation. A 2D dot product look-up operation takes three textures in one pass. Without loss of generality, we assume that the first three input textures are used. In Figure 8.6, the parameter $\mathbf{U}_x$ is the dot product of $tex0$ and the texture coordinates of $tex1$ (texcoord($tex1$)) while the parameter $\mathbf{U}_y$ is the dot product of $tex0$ and the texture coordinates of $tex2$ (texcoord($tex2$)). They are used to look up the texture defined in the input texture $tex2$. Texture coordinates can be defined by specifying four coordinates for the corners of the input texture. Texture coordinates for other pixels in the texture are calculated by interpolating these four coordinates. If we define four identical coordinates, then all texture coordinates of pixels in the texture will be the same. After the 2D dot product look-up operation, the output texture is the looked up texture $tex2(\mathbf{U}_x, \mathbf{U}_y)$. We shall discuss the usefulness of this operation shortly.

After texture shader operations, the output textures are passed to the register combiner. The operations of register combiner are mainly for blending textures. In other words, given the output textures, the register combiner applies arithmetics among the output textures. For example, given two output textures (for simplicity, assumed the textures are in gray scale),

$$\begin{bmatrix} 0.1 & 0.7 \\ 0.2 & 0.4 \end{bmatrix} \textit{and} \begin{bmatrix} 0.3 & 0.2 \\ 0.2 & 0.5 \end{bmatrix},$$

if we apply addition on them in the register combiner, we will get

$$\begin{bmatrix} 0.4 & 0.9 \\ 0.4 & 0.9 \end{bmatrix}.$$

If we apply multiplication on them, we will get

$$\begin{bmatrix} 0.03 & 0.14 \\ 0.04 & 0.2 \end{bmatrix}.$$

More complex arithmetics can also be done on the textures in the register combiner. However, in our implementation, addition and multiplication are more than enough for our purpose.

Relighting using Pixel Shader

By using the operations (2D/rectangular textures, 2D dot product look-up, and texture blending) just introduced, we can re-formulate our relighting problem such that the relighting process can be achieved in real-time. If we store the eigenimages and relighting coefficients in the input textures, what we want to do is a linear combination of them (Figure 8.7). However, there are limitations and other problems that have to be solved.

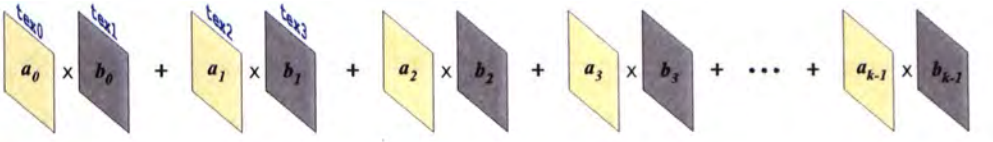


Figure 8.7: Linear combination of textures.

In order to facilitate the shader operations, we have to expand the relighting coefficients to the same size as the eigenimages (Figure 8.8). In other words, as one coefficient is responsible for one block of the eigenimages, each coefficient needs to grow to a size same as the block size (i.e. a block with size 16×16 having its coefficients all equal to the same value). By expanding each set of relighting coefficients, the shader simply needs to multiply the two textures in the register combiner. However, if we expand it in the first-phase decoding, memory is wasted due to having too many repeated values. We can use 2D dot product look-up operation for the expansion. We shall discuss it shortly.

In Section 8.3.1, we have mentioned that interpolation is needed for relighting images with lighting directions that do not appear in the samples. We have to interpolate the four sets of relighting coefficients. However, by using

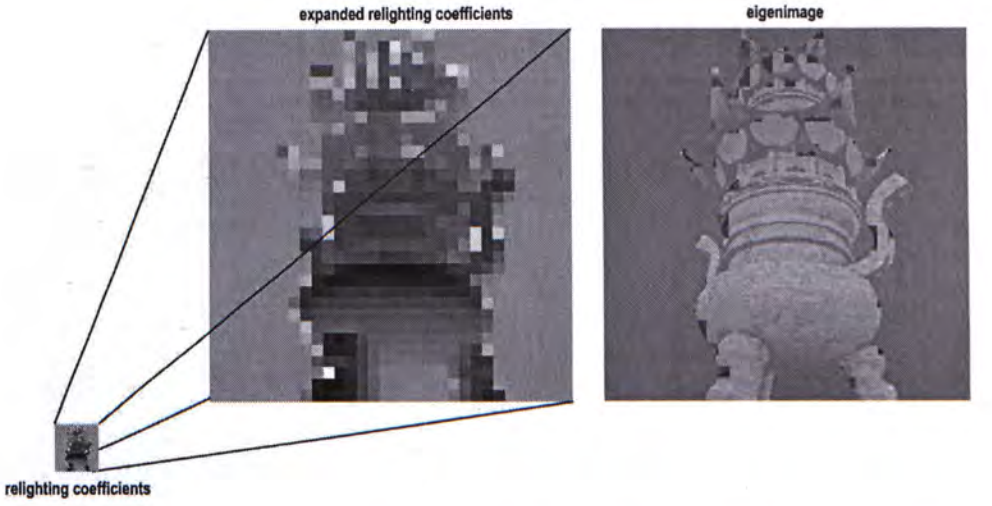


Figure 8.8: The relighting coefficients are expanded to the same size as the eigenimages.

nVidia GeForce 3, we can leave the interpolation work to the shaders. As the interpolation in the shaders can only be done on adjacent pixels, we have to rebin the relighting coefficients such that each sub-block contains the relighting coefficients corresponding to a block of the corresponding eigenimages. The rebinned first set of relighting coefficients for the Y channel of the data set 'ding' is shown in Figure 8.9. It is not surprising that a rough appearance of the 'ding' can be outlined, as each pixel in a 'relight' block ($p \times q$) here corresponds to the same 'image' block (16×16) of the eigenimages. Hereafter, we distinguish these two kinds of blocks as 'image' block and 'relight' block. The rebin process should be carried out in the first-phase decoding in order to save the relighting time.

The expansion and interpolation problem mentioned above (that is, for the pixels within the same 'image' block of the output texture, the same relighting coefficient in $tex2$ is being looked up) can be achieved simultaneously by using 2D dot product look-up operation. If we view this problem in a mathematical sense, for example, if the 'image' block size we use is 2×2 and the image size

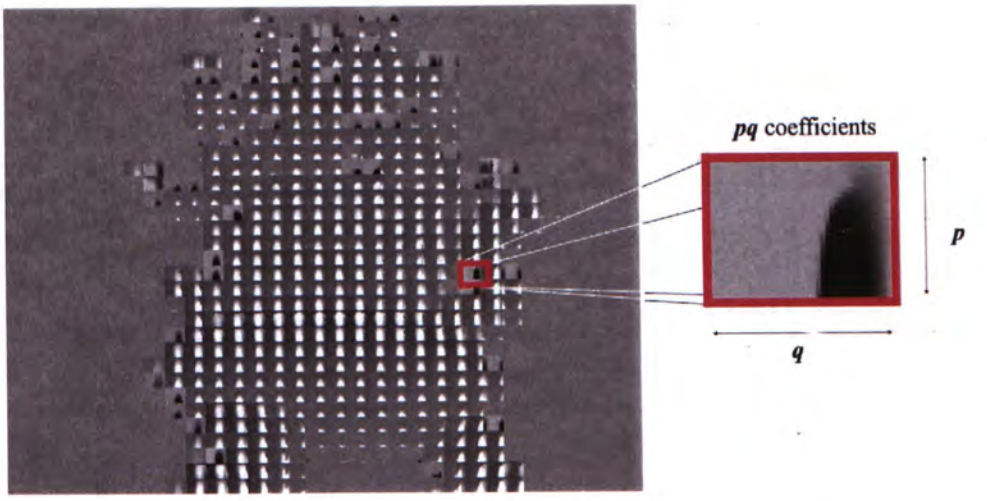


Figure 8.9: Coefficient arrangements for hardware relighting.

is 6×6, what the relighting coefficients we hope to have are

$$\begin{bmatrix}
tex2(p,t) & tex2(p,t) & tex2(\phi+p,t) & tex2(\phi+p,t) & tex2(2\phi+p,t) & tex2(2\phi+p,t) \\
tex2(p,t) & tex2(p,t) & tex2(\phi+p,t) & tex2(\phi+p,t) & tex2(2\phi+p,t) & tex2(2\phi+p,t) \\
tex2(p,\theta+t) & tex2(p,\theta+t) & tex2(\phi+p,\theta+t) & tex2(\phi+p,\theta+t) & tex2(2\phi+p,\theta+t) & tex2(2\phi+p,\theta+t) \\
tex2(p,\theta+t) & tex2(p,\theta+t) & tex2(\phi+p,\theta+t) & tex2(\phi+p,\theta+t) & tex2(2\phi+p,\theta+t) & tex2(2\phi+p,\theta+t) \\
tex2(p,2\theta+t) & tex2(p,2\theta+t) & tex2(\phi+p,2\theta+t) & tex2(\phi+p,2\theta+t) & tex2(2\phi+p,2\theta+t) & tex2(2\phi+p,2\theta+t) \\
tex2(p,2\theta+t) & tex2(p,2\theta+t) & tex2(\phi+p,2\theta+t) & tex2(\phi+p,2\theta+t) & tex2(2\phi+p,2\theta+t) & tex2(2\phi+p,2\theta+t)
\end{bmatrix}$$

where (p,t) is the desired lighting direction and $0 \leq p \leq \phi$ and $0 \leq t \leq \theta$. In other words, we are trying to set $\mathbf{U}_x$ and $\mathbf{U}_y$ to be

$$\mathbf{U}_x = \begin{bmatrix}
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \\
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \\
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \\
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \\
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \\
p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p
\end{bmatrix} \textit{ and}$$

$$\mathbf{U}_y = \begin{bmatrix}
t & t & t & t & t & t \\
t & t & t & t & t & t \\
\theta+t & \theta+t & \theta+t & \theta+t & \theta+t & \theta+t \\
\theta+t & \theta+t & \theta+t & \theta+t & \theta+t & \theta+t \\
2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t \\
2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t & 2\theta+t
\end{bmatrix}.$$

Actually, they can be reduced to

$$\mathbf{U}_x = \begin{bmatrix} p & p & \phi+p & \phi+p & 2\phi+p & 2\phi+p \end{bmatrix} \textit{ and}$$

$$\mathbf{U}_y = \begin{bmatrix}
t \\ t \\ \theta+t \\ \theta+t \\ 2\theta+t \\ 2\theta+t
\end{bmatrix}.$$

So, for pixel of the output texture (expanded relighting coefficients) at (i, j), we should look up the pixel of the input texture $tex2$ at $(\mathbf{U}_x(i,j), \mathbf{U}_y(i,j))$ or simply $(\mathbf{U}_x(i), \mathbf{U}_y(j))$, where the texture $tex2$ stores the rebinned relighting coefficients. Note that the coordinate (i, j) here means column i and row j.

In order to set $\mathbf{U}_x$ and $\mathbf{U}_y$, we have to define an offset texture which is stored in $tex0$. Using the same example ('image' block size is 2×2 and the image size is 6×6), then we should have the offset texture,

$$\begin{bmatrix} (0,0,1) & (0,0,1) & (1,0,1) & (1,0,1) & (2,0,1) & (2,0,1) \\ (0,0,1) & (0,0,1) & (1,0,1) & (1,0,1) & (2,0,1) & (2,0,1) \\ (0,1,1) & (0,1,1) & (1,1,1) & (1,1,1) & (2,1,1) & (2,1,1) \\ (0,1,1) & (0,1,1) & (1,1,1) & (1,1,1) & (2,1,1) & (2,1,1) \\ (0,2,1) & (0,2,1) & (1,2,1) & (1,2,1) & (2,2,1) & (2,2,1) \\ (0,2,1) & (0,2,1) & (1,2,1) & (1,2,1) & (2,2,1) & (2,2,1) \end{bmatrix}.$$

Each element in the texture is a tuple because it is a color pixel (3 channels). And if we fix texture coordinates of all the pixels of $tex1$ and $tex2$ to be constants $(\phi, 0, p)$ and $(0, \theta, t)$, then after applying the 2D dot product look-up operation in the texture shader, we shall get the $\mathbf{U}_x$ and $\mathbf{U}_y$ that are with the desired values mentioned above. Take the element $(2, 2, 1)$ in the offset texture as example, we will have the corresponding elements in $\mathbf{U}_x$ and $\mathbf{U}_y$ to be $(2,2,1).(\phi, 0, p) = 2\phi + p$ and $(2,2,1).(0, \theta, t) = 2\theta + t$ respectively.

In summary, by using the 2D dot product look-up operation in the texture shader with the offset texture in $tex0$ and the relighting coefficients in $tex2$ and texture coordinates of all the pixels of $tex1$ and $tex2$ to be constants, we expand each relighting coefficient stored in $tex2$ to the size same as the 'image' block size (2×2 in the example) and each set of the relighting coefficients to the size same as the eigenimage (6×6 in the example). To generalize, the offset texture should have the same size as the eigenimages. Values in each block of the offset texture are all equal to the "block" coordinate. Figure 8.10 illustrates the idea.

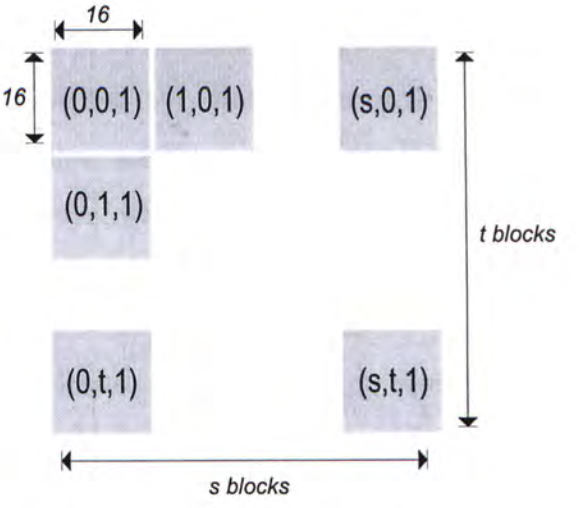


Figure 8.10: Generalized offset texture.

We have used 3 textures for expanding and interpolating the relighting coefficients. The only texture left can be used for storing an eigenimage so that we can multiply the relighting coefficients with an eigenimage in the register combiner in one shader pass. The corresponding register combiner code is as follow,

```
---------------------------------
 1. !!RC1.0
 2. const0 = (1.0, 1.0, 1.0, 0.0);
 3. { rgb{ spare1 = tex2.a*tex3; } }
 4. { rgb{
 5.      discard = const0;
 6.      discard = spare1;
 7.      spare0 = sum();
 8.      scale_by_one_half(); } }
 9. out.rgb = spare0;
10. out.a = unsigned_invert(zero);
---------------------------------
```

Register combiner codes look like assembly codes. Line 1 of the code above indicates that this is a piece of codes for the register combiner. Line 2 tries to set a constant $(1.0, 1.0, 1.0, 0.0)$. Each outermost pair of braces encloses one stage of register combiner actions (remember that we have at most 8 stages of register combiner actions, which are carried out in series). Line 3 defines one stage of register combiner actions, which tries to multiply the input textures $tex2$ and $tex3$, where $tex2$ stores the expanded and interpolated relighting coefficients and $tex3$ stores the corresponding eigenimage. The token $tex2.a$ means the alpha channel of the input texture $tex2$. Let's recall that all three channels of an pixel share the same relighting coefficient. We can save space if we only use the alpha channel. The product is stored in the temporary register $spare1$. The second stage of register combiner actions tries to scale the values in $spare1$ to the range [0,1] (add 1's, the constant $const0$, to $spare1$ and divided the sum by two). It is because the product is in the range [-1,1] while the pixel shader will clamp the output to the range [0,1]. Line 9 and 10 set the channels (RGB and alpha respectively) of the output. The function $unsigned\_invert(zero)$ is simply 1 because there is no 'one' but 'zero' defined in the shader.

After obtaining the products of the relighting coefficients and the eigenimages, we have to add them up to complete the linear combination. This can be easily done by setting the textures in the texture shader as the products and adding them up using the register combiner. The linear combination takes several shader passes, therefore the quality of the relit images is affected due to precision loss (precision of the shaders is limited to 8 bits when our system was being built). Moreover, there is trade-off to utilize current commodity hardware. Besides the precision of current graphics hardware is limited to 8-bit per pixel, all computations must be taken within the range of [0,1]. Even though we distribute energy evenly to matrices $\widehat{\mathbf{A}}$ and $\widehat{\mathbf{B}}$ during SVD (Section 5.2), error still exists. The evaluation result will be given in Chapter 9.

Chapter 9

Overall Evaluation

In this chapter, we evaluate the compression method we proposed as well as the relighting engine we implemented. We compare the performances of using M-JPEG, MPEG, spherical harmonics, and our method for compressing illumination-adjustable images in terms of reconstruction PSNR. For hardware-assisted relighting, we evaluate the reconstruction PSNR as well as the relighting speed in terms of frame rate.

9.1 Compression of IAIs

9.1.1 Statistical Evaluation

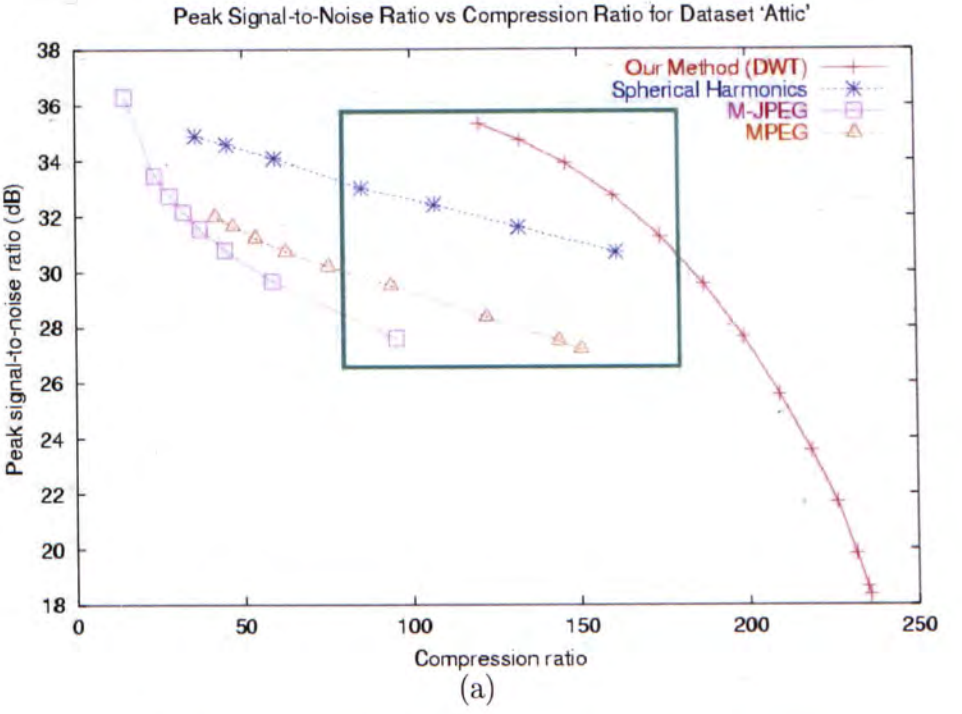
To evaluate the overall performance of the proposed PCA-based encoding method, we compare it to other video encoding methods. Just like video compression, we also compress multiple images. Therefore, it is more appropriate to compare our method to video compression methods. However, there is fundamental difference between our image-based relighting application and video playback. Video playback is usually 1D (either forward or backward playback) while relighting is 2D as the user moves the light source on a spherical surface. In addition to video compression methods, we also compare to method that is tailor-made for compressing illumination-adjustable images [10]. The

comparison results reveal that our method out-performs the video compression methods and the tailor-made method.

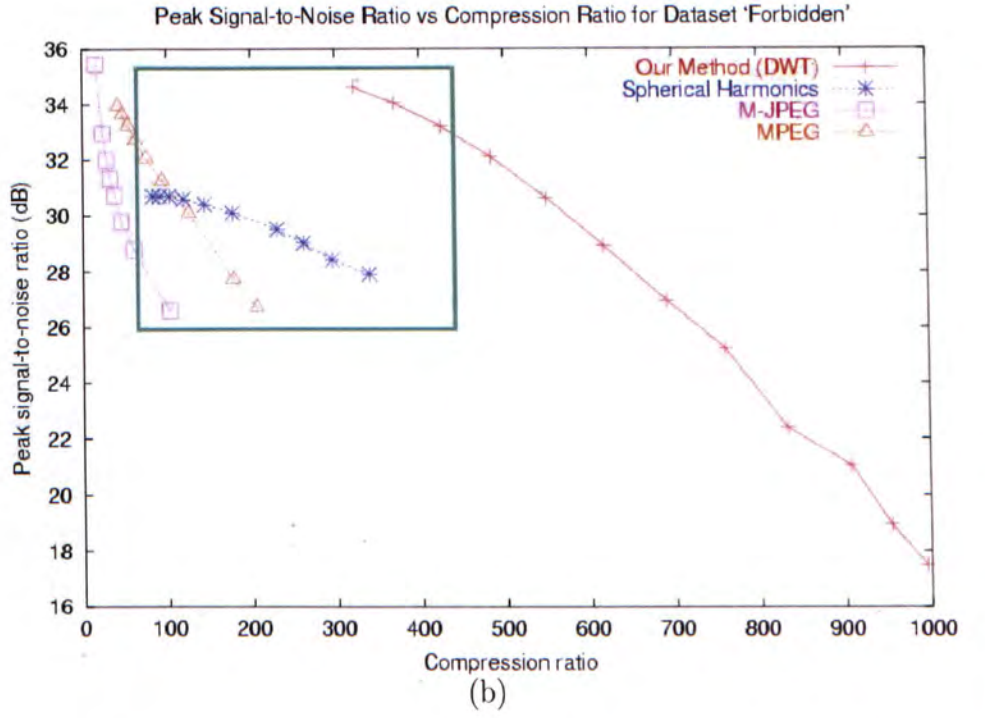
We use the setting suggested in previous sections ($k = 9$, $R_A = 4.0$, and $R_B = 3.0$ for DCT while $k = 9$, $R_A = 4.0$, and $R_B = 1.0$ for DWT) to compress the four data sets. The compression ratios and reconstruction errors are then measured and recorded. The same four data sets are then compressed using two video coding methods, Motion JPEG (M-JPEG) and MPEG, and the tailor-made method, Spherical Harmonics. We want to determine the compression ratios of them given the same image quality (in terms of PSNR).

Since we cannot directly control the desired image quality during video compression and the tailor-made method, we can only estimate the compression ratio in the following manner. For each coding method and each data set, we compress the data set using different target bit rates. The corresponding reconstruction errors are measured. Then a graph of reconstruction error against the target bit rate is plotted and the compression ratio giving the 'same' image quality is estimated from this graph. The graphs for data sets 'attic' and 'forbidden' are shown in Figures 9.1. Not all parts of the graphs are of interest because we do not compress images with bad reconstruction quality (both visually and statistically) in practice. The same is for compression ratio, we do not compress images with too small compression ratio. Therefore, we highlight the areas (enclosed by the green rectangles) that are of interest in the graphs. Using this method, we tabulate the compression ratios of M-JPEG, MPEG, and Spherical Harmonics (S.H.) in Table 9.1. Because we can only obtain the results of compressing the data sets 'attic' and 'forbidden' using the spherical harmonics method, the results of compressing the data sets 'ding' and 'cover' are missing.

In every case (excluding the one using DCT in compressing eigenimages for the data set 'attic'), the proposed method out-performs the other methods. It



(a)



(b)

Figure 9.1: PSNR versus compression ratio for Spherical Harmonics using data sets (a) 'attic' and (b) 'forbidden'.

Data set	PSNR (dB)	M-JPEG	MPEG	S.H.	Our method
ding	36.9	50.5	45.0	n/a	538.9
attic	31.0	42.5	57.4	150.3	97.8
cover	39.1	42.0	43.0	n/a	575.2
forbidden	33.2	22.9	54.7	103.0	373.4

(a) Using DCT in eigenimage coding

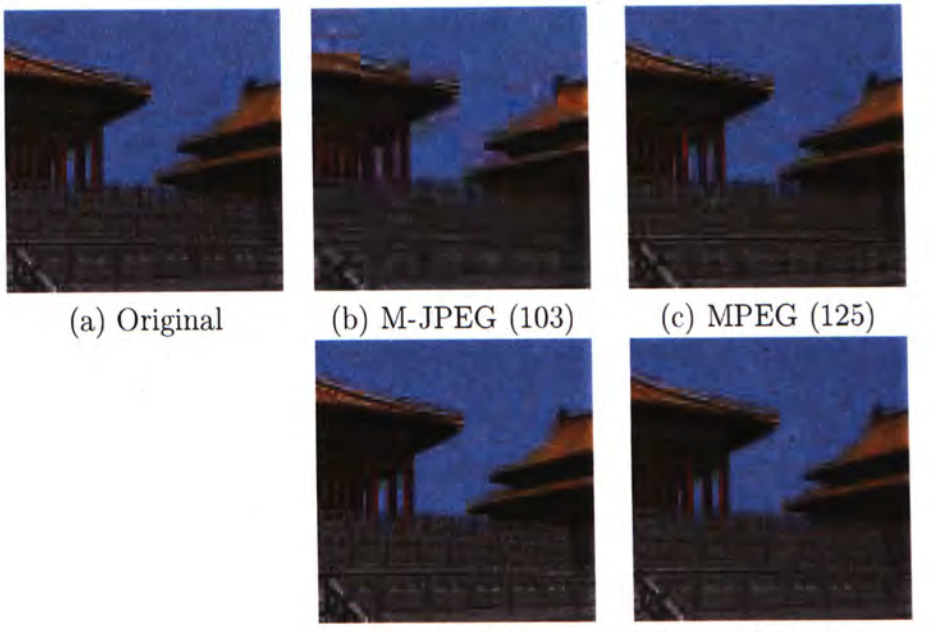
Data set	PSNR (dB)	M-JPEG	MPEG	S.H.	Our method
ding	37.4	48.5	44.0	n/a	756.3
attic	30.8	44.6	61.3	157.3	174.4
cover	39.5	38.2	42.0	n/a	737.3
forbidden	32.1	28.0	74.7	103.0	483.9

(b) Using DWT in eigenimage coding

Table 9.1: Comparison of compression ratios of different encoding methods.

is expected that M-JPEG is the worst because it does not exploit the coherence among images. But our method also out-performs MPEG. The result may be due to the fact that MPEG can only exploit coherence among the two consecutive frames (1D) while our method can utilize coherence in 2D. It is also shown that for all data sets, using DWT in eigenimage coding can give a higher compression ratio while maintaining almost the same relighting quality. On the other hand, it is expected that the tailor-made method, that is the S.H. method, performs better than M-JPEG and MPEG. By using spherical harmonics, with PSNR's around 31dB and 32dB, compression ratios for data sets 'attic' and 'forbidden' are about 150 and 103 respectively. However, by using our method, the corresponding compression ratios are 174 and 484 respectively. Referring to Table 9.1, the S.H. method performs better in compressing the data set 'attic' if we use DCT in our method, but it does not perform better than our method using DWT. Moreover, we should use DWT in compressing the eigenimages as it is proved in the previous chapter (Section 6.2) that DWT performs better than DCT does. In other words, our method also out-performs the S.H. method, which is tailor-made for compressing illumination-adjustable images, especially for the data set 'forbidden'.

9.1.2 Visual Evaluation



(a) Original (b) M-JPEG (103) (c) MPEG (125)

(c) S.H. (120) (d) Our Method (322)

Figure 9.2: Visual comparison for all the compression methods. The numbers in the brackets are the compression ratios.

Besides statistical evaluation, we also have a visual evaluation. Figure 9.2 shows reconstructed images from the compressed data using different methods for the data set 'forbidden'. The compression ratios are 103, 125, 120, and 322 for M-JPEG, MPEG, S.H., and our method respectively. M-JPEG performs the worst among all the compression methods. Severe visual artifacts appear in the reconstructed image. MPEG performs better than M-JPEG. However, the fences are still blurred. The detail of them cannot be seen. The S.H. method and our method perform the best. The details of the fences can be seen clearly. Comparing to the original image, all three images are almost identical visually. However, with almost the same visual quality, our method can have a compression ratio 322 while the S.H. can only have 120. In other words, our method out-performs the S.H. method. Therefore, our method

out-performs all the three methods in this visual evaluation.

In summary, our method out-performs all of the above methods both statiscally and visually. Moreover, our proposed method not just effectively compresses data but also facilitates the user retrieval pattern (2D) in our relighting application.

9.2 Hardware-Assisted Relighting

To evaluate the performance of hardware-assisted relighting, we measure the relighting speed and quality of the relit images.

To measure the relighting speed, we cast 1000 light sources and ask the system to relight the images. We use the Hammersley points [52] to generate the lighting directions. Figure 9.3 shows 1000 Hammersley points on a sphere. By using this method, we can avoid overcrowding some of the samples, which is a problem if we use random numbers. The time required to finish relighting the 1000 relit images is recorded.

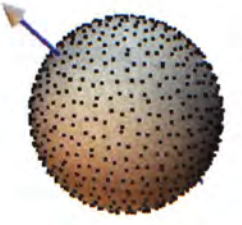


Figure 9.3: Hammersley points on sphere with 1000 samples.

Table 9.2 shows that the relighting speed of our data sets is in real-time (we use 9 eigenimages for each case). All data sets can reach 18 or above frames per second. Actually, the relighting speed mainly depends on two parameters, namely number of eigenimages used (i.e. number of shader passes) and image resolution. In our experiment, all data sets use 9 eigenimages, therefore only

the image resolution varies. The data set 'cover' has a smaller resolution, that is why it has a faster relighting speed. Although the shader is claimed to process the pixels in parallel, we believe that the shader has a limit on the image resolution, therefore image resolution still affects the relighting speed.

Data set	frame rate (fps)	PSNR (dB)
ding	18.9	27.9
attic	18.9	28.9
cover	76.9	27.9
forbidden	18.9	28.7

Table 9.2: Frame rate and error measurement of hardware-accelerated relighting.

To measure the relighting quality, we reconstruct m images from *uncompressed* $\widehat{\mathbf{A}}$ and *uncompressed* $\widehat{\mathbf{B}}$. The reconstructed images are compared to the same control images in Section 6.2. By this, we can measure the errors solely due to the hardware. Table 9.2 shows the results. Although the precision in the shader is limited to 8-bit per pixel, the PSNR for each data set is around 28. If the precision of the next generation shaders increases, the reconstruction relighting quality will become much better in the future. Figure 9.4 shows the visual artifacts due to hardware relighting. The artifacts are mainly due to the truncations of data (relighting coefficients) that exceed the range $[-1, 1]$.

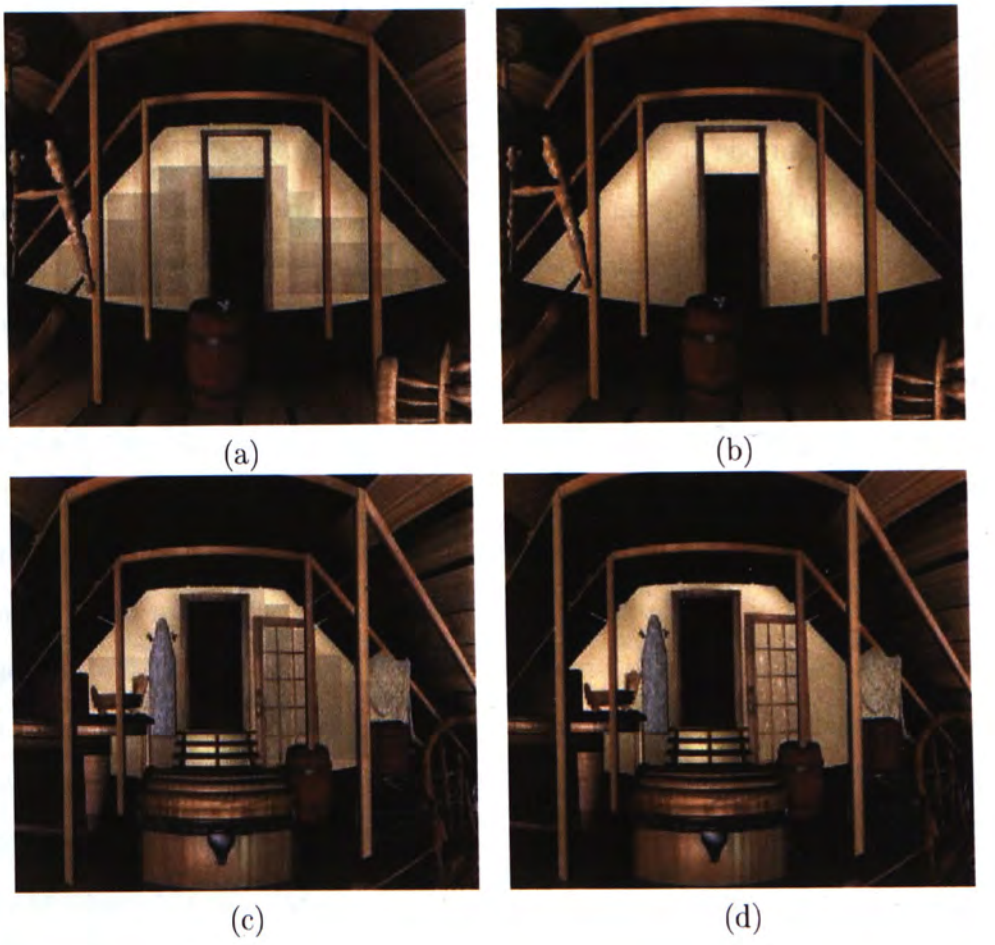


Figure 9.4: The visual artifacts due to hardware relighting. (a) and (c): visual artifacts. (b) and (d): the corresponding relit images using software relighting.

Chapter 10

Conclusion

In this thesis, we proposed a block-wise PCA-based method for compressing image-based relighting data. Since the data volume is enormous, we divide-and-conquer the problem in order to make it manageable. The eigenimages and relighting coefficients are compressed separately according to their data natures. A high compression ratio is achieved (Table 9.1) while maintaining the visual quality. It out-performs standard video coding method as it exploits the data correlation specific to image-based relighting. To facilitate real-time relighting, we utilize consumer-level graphics hardware to synthesize the desired images. The real-time performance also demonstrates the advantage of image-based approach. Unfortunately, current hardware introduces error to the synthesized images due to the limited data precision. The problem will be solved once the next-generation high-precision graphics hardware becomes available. Currently we store equal number of eigenimages for each block. The compression ratio can be further increased if the number of eigenimages adapts to the image content. For instance, constant background may require less number of eigenimages. Moreover, our proposed compression scheme is generic for compressing image-based data. It can also compress other image-based data such as Light field. However, as our proposed method is tailor-made for compressing illumination-adjustable images, further investigation must be made in prior to applying it on other image-based data.

Bibliography

[1] Tien Tsin Wong, Pheng Ann Heng, Siu Hang Or, and Wai Yin Ng, "Image-based rendering with controllable illumination," in *Eighth Eurographics Workshop on Rendering*, June 1997, pp. 13–22.

[2] Yizhou Yu and Jitendra Malik, "Recovering photometric properties of architectural scenes from photographs," in *SIGGRAPH '98 Conference Proceedings*. ACM SIGGRAPH, July 1998, Annual Conference Series.

[3] Tien-Tsin Wong, Chi-Wing Fu, and Pheng-Ann Heng, "Interactive relighting of panoramas," *IEEE Computer Graphics & Applications*, vol. 21, no. 2, pp. 32–41, March-April 2001.

[4] Marc Levoy and Pat Hanrahan, "Light field rendering," in *SIGGRAPH*, August 1996, pp. 31–42.

[5] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen, "The lumigraph," in *SIGGRAPH*, August 1996, pp. 43–54.

[6] Heung Yeung Shum and Li Wei He, "Rendering with concentric mosaics," in *SIGGRAPH*, August 1999, pp. 299–306.

[7] Shenchang Eric Chen, "QuickTime VR - an image-based approach to virtual environment navigation," in Computer Graphics *Proceedings, Annual Conference Series, SIGGRAPH'95*, August 1995, pp. 29–38.

[8] Marcus Magnor and Bernd Girod, "Data compression for light-field rendering," in *IEEE transactions on CSVT*, April 2000, vol. 10, pp. 338–343.

[9] Chandrajit Bajaj, Insung Ihm, and Sanghun Park, "3D RGB image compression for interactive applications," *ACM Transactions on Graphics*, vol. 20, pp. 10–38, January 2001.

[10] Tien-Tsin Wong, Chi-Wing Fu, Pheng-Ann Heng, and Chi-Sing Leung, "The plenoptic illumination function," *IEEE Transactions on Multimedia*, vol. 4, no. 3, September 2002.

[11] Peter N. Belhumeur and David J. Kriegman, "What is the set of images of an object under all possible lighting conditions?," in *IEEE Conference on Computer Vision and Pattern Recognition*, 1996, pp. 270–277.

[12] Russell Epstein, Peter W. Hallinan, and Alan L. Yuille, "5+/-2 eigenimages suffice: An empirical investigation of low-dimensional lighting models," in *Proceedings of IEEE Workshop on Physics-Based Modeling in Computer Vision*, Cambridge, Massachusett, June 1995, pp. 108–116.

[13] Zhengyou Zhang, "Modeling geometric structure and illumination variation of a scene from real images," in *Internation Conference on Computer Vision*, January 1998.

[14] Ko Nishino, Yoichi Sato, and Katsushi Ikeuchi, "Eigen-texture method: Appearance compression based on 3D model," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 1999, vol. 1, pp. 618–624.

[15] Pun-Mo Ho, Tien-Tsin Wong, and Chi-Sing Leung, "PCA-based compression for image-based relighting," *Submitted to IEEE International Conference on Multimedia and Expo*, July 2003.

[16] Pun-Mo Ho, Tien-Tsin Wong, and Chi-Sing Leung, "Compressing the illumination-adjustable images with principal component analysis," *Submitted to IEEE Transactions on Circuits and Systems for Video Technology*, 2003.

[17] Jin-Xiang Chai, Xin Tong, Shing-Chow Chan, and Heung-Yeung Shum, "Plenoptic sampling," in *Computer Graphics (SIGGRAPH 2000 Proceedings)*, July 2000.

[18] Marcus Magnor and Bernd Girod, "Hierarchical coding of light fields with disparity maps," in *ICIP*, October 1999.

[19] Cha Zhang and Jin Li, "Compression of lumigraph with multiple reference frame (mrf) prediction and just-in-time rendering," in *IEEE Conference on Data Compression*, 2000, pp. 253–262.

[20] Ingmar Peter and Wolfgang Straber, "The wavelet stream - progressive transmission of compressed light field data," in *IEEE Visualization*, 1999.

[21] Insung Ihm, Sanghoon Park, and Rae Kyoung Lee, "Rendering of spherical light fields," in *Pacific Graphics*, 1997.

[22] Gavin Miller, Steven Rubin, and Dulce Ponceleon, "Lazy decompression of surface light fields for precomputed global illumination," in *Proccedings of the Eurographics Workshop*, 1998.

[23] D.N. Wood, D.I. Azuma, K. Aldinger, B. Curless, T. Duchamp, D.H. Salesin, and W. Stuetzle, "Surface light fields for 3D photographs," in *SIGGRAPH*, 2000.

[24] W.C. Chen, J.Y. Bouguet, M.H. Chu, and R. Grzeszczuk, "Light field mapping: Efficient representation and hardware rendering of surface light fields," in *SIGGRAPH*, 2002.

[25] Y. Wu, C. Zhang, J. Li, and J. Xu, "Smart-rebinning for compression of concentric mosaics," in *ACM Multimedia*, 2000.

[26] Wing Ho Leung and Tsuhan Chen, "Compression with mosaic prediction for image-based rendering applications," in *IEEE International Conference on Multimedia and Expo*, 2000, vol. 3, pp. 1649–1652.

[27] Jin Li, Heung Yeung Shum, and Ya Qin Zhang, "On the compression of image based rendering scene," *International Journal of Image and Graphics*, vol. 1, no. 1, pp. 45–61, 2001.

[28] Jeffry S. Nimeroff, Eero Simoncelli, and Julie Dorsey, "Efficient re-rendering of naturally illuminated environments," in *Fifth Eurographics Workshop on Rendering*, June 1994, pp. 359–373.

[29] R. Courant and D. Hilbert, *Methods of Mathematical Physics*, Interscience Publisher, Inc, 1953.

[30] Edward H. Adelson and James R. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*, Michael S. Landy and J. Anthony Movshon, Eds., chapter 1, pp. 3–20. MIT Press, 1991.

[31] A. Gershun, "The light field," *Journal of Mathematics and Physics*, vol. XVIII, pp. 51–151, 1939, Translated by P. Moon and G. Timoshenko.

[32] Wai-Man Pang, "A portable capturing system for image-based relighting," M.Phil. Thesis, Department of Computer Science, the Chinese University of Hong Kong, July 2002.

[33] G. K. Wallace, "The JPEG still image compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, April 1991.

[34] Didier Le Gall, "MPEG: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46–58, April 1991.

[35] Eric Hamilton, "JPEG file interchange format, version 1.02," http://www.jpeg.org/public/jfif.pdf, September 1992.

[36] C.S. McGoldrick, W.J. Dowling, and A. Bury, "Image coding using the singular value decomposition and vector quantization," in *Fifth International Conference on Image Processing and its Applications*, 1995, pp. 296–300.

[37] G.W. Stewart, "On the early history of the singular value decomposition," Tech. Rep. CS-TR-2855, Department of Computer Science, University of Maryland, College Park, 1992.

[38] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1989.

[39] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1993.

[40] Michael Berry, Theresa Do, Gavin O'Brien Vijay Krishna, and Sowmini Varadhan, "Svdpackc (version 1.0) user's guide," .

[41] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Addison-Wesley, 1993.

[42] R. M. Gray, *Source Coding Theory*, Kluwer Academic, 1990.

[43] N.S. Jayant and P. Noll, *Digital Coding of Waveforms: Principles and Applications to Speech and Video*, Prentice-Hall, 1984.

[44] R.C. Reininger and J.D. Gibson, "Distribution of the two-dimensional dct coefficients for images," *IEEE Transactions on Communications*, vol. COM-31, pp. 835–839, 1983.

[45] A. N. Akansu and M. J. T. Smith, *Multiresolution Signal Decomposition: Transforms, Subands, and Wavelets*, Academic Press, 1992.

[46] R. L. Joshi, H. Jafarkani, J. H. Kasner, T. R. Fischer, N. Farvardin, M. W. Marcellin, and R. H. Bamberger, "Comparison of different methods of classification in subband coding of images," *IEEE Transactions on Image Processing*, vol. 6, pp. 1473–1486, 1997.

[47] A.M. Tekalp, *Digital Video Processing*, Prentice-Hall, 1998.

[48] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi, "The jpeg 2000 still image compression standard," September 2001.

[49] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," in *IEEE Transactions on Signal Processing*, Dec 1993, vol. 4.

[50] R.W. Buccigrossi and E.P. Simoncelli, "Image compression via joint statistical characterization in the wavelet domain," in *IEEE Transactions on Image Processing*, December 1999.

[51] Erik Lindholm, Mark J. Kilgard, and Henry Moreton, "A user-programmable vertex engine," in *Proceedings of SIGGRAPH 2001*, August 2001, pp. 149–158.

[52] T.T. Wong, W.S. Luk, and P.A. Heng, "Sampling with Hammersley and Halton points," in *Journal of Graphics Tools*, 1997, vol. 2.