EE398A: Group 2 Project Report

Christopher Li, Idoia Ochoa and Nima Soltani
{chriswli, iochoa, nsoltani}@stanford.edu

***Abstract*—We present an approach to compress a stereo image, which comprises a left and a right image, both of which are typically correlated with each other. To minimize the rate, in bits per pixel, of the compressed image, we incorporate several common encoding techniques found in JPEG-2000 with residual image coding and quantization step search. Results show that our scheme is able to compress the image pair while achieving a specified target PSNR at the reconstructed image.**

I. Introduction

While JPEG-2000 is the leading standard for image compression, it does not exploit similarities between the two images in a stereo image pair to achieve a further reduction in the amount of bits needed. In this project, we adapt various features of JPEG-2000, such as block coding and zig-zag scanning [1] to develop a compression scheme for stereo images, which are of interest because they can provide a 3-D representation of a scene [2]. Fig. 1 is an example of a stereo image pair.

If we can exploit similarities between the two images, we can reduce the amount of information that is needed to encode the two images. While both images in the stereo pair have similar visual properties, there are inherent differences between the two images. In particular, there may be a spatial offset between a feature in the left image and the same feature in the right image, such that the right image can be considered a translated copy of the left image. However, the relationship between the two images may be more complicated than that of a one-dimensional translation between the two images. The challenge is to identify and exploit similarities of the image while meeting a minimum requirement on the quality of the reconstructed image following decompression.

Fig. 1: Example of stereo image

II. Problem Description

The aim of this project is to construct an encoding-decoding pair for stereo images (see Fig. 1) while satisfying a PSNR requirement of 37 dB and an average encoding-decoding time of two minutes. The images are composed of a left and a right image that are highly correlated. Each image is 540×960 pixels in size, and is decomposed into its luminance (Y) and chrominance components (C_b and C_r), which are then encoded separately. The remaining components are then down sampled by 2, leading to an image of dimension 270×480 pixels. MATLAB is the main programming language used for this project. Hereafter, the subscripts $L_Y, R_Y, L_{C_b}, R_{C_r}, L_{C_r}$ and R_{C_r} denote the Y, C_b and C_r components for the left and right images, respectively.

The performance of the encoder is measured by the compression rate, as measured in bits per pixel.

III. System Overview

The encoder scheme is illustrated in Fig. 2. The left image is encoded separately by using a discrete wavelet transform followed by a uniform quantizer, which is optimized to find the biggest step size while meeting a require MSE, as explained in the next section. The resulting quantized version is then encoded by an arithmetic coder. For the right image, we decide between encoding it separately, intra mode, or with the help of the left image as side-information, inter mode. The decision is based on the MSE incurred by L_Y after the quantization. If it is bigger than a specific parameter, we choose the intra mode, and the inter mode otherwise. When R is encoded separately, the same steps as for the left image are performed. For the differential encoding, we calculate the motion vectors and the residuals, and then encode them using Huffman and arithmetic coding, respectively. Once the decision on whether or not to code the right image individually is made, all of the blocks in the right image follow the same coding scheme. The main reason why we do not consider a block decision scheme is that in stereo images, as opposed to video, we know both left and right images are typically highly correlated.

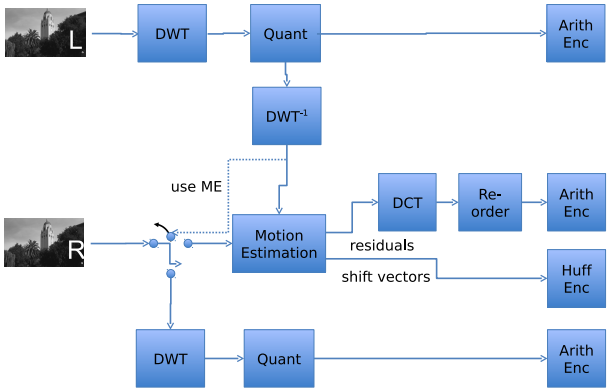

Fig. 2: System Overview

IV. Quantization level search

We want to ensure a minimum PSNR between the original image and its lossy version, where the PSNR is given by

$$PSNR = 10 \log \frac{255^2}{MSE} \text{ [dB]} \quad (1)$$

Throughout the paper, $k \in \{Y, C_b, C_r\}$ denotes the image component. We denote by L_k and R_k the original left and right components, respectively, and by $\hat{L}_k$ and $\hat{R}_k$ the reconstructed versions. w_k and h_k stand for the width and height of the k component, respectively. For ease of notation, we define

$$MSE_L = \sum_k MSE_{L_k} = \sum_k \sum_{i=1}^{h_k} \sum_{j=1}^{w_k} (L_k - \hat{L}_k)_{i,j}^2$$

$$MSE_R = \sum_k MSE_{R_k} = \sum_k \sum_{i=1}^{h_k} \sum_{j=1}^{w_k} (R_k - \hat{R}_k)_{i,j}^2$$

The MSE is computed as

$$MSE = \frac{MSE_L + MSE_R}{2 \cdot 1.5 \cdot w_Y \cdot h_Y} \quad (2)$$

Combining the above equations (1) and (2), we have that to meet the given PSNR requirement we must satisfy

$$MSE_L + MSE_R \leq \frac{2 \cdot 1.5 \cdot w_Y \cdot h_Y \cdot 255^2}{10^{PSNR/10}} \quad (3)$$

We force the encoder to allocate the same MSE for the left and the right image, i.e. $MSE_L = MSE_R$. Any other combination would work, but we want both of them to have the same MSE, so that the distortion induced in the decoded image is the same for both left and right parts.

Denote the right part of equation (3) as MSE^*. Then, we find the biggest the step size Δ_k such that $MSE_k = a_k \frac{MSE^*}{2}$, where the a_k's are chosen to satisfy $a_{C_b} + a_{C_r} + a_Y = 1$. This method ensures the final PSNR to be as close to the target PSNR of 37 dB.

We allocate 8 bits for each Δ_k, which means that our quantizer looks for the maximum $\Delta_k \in 1, \ldots, 255$. To speed up the computation, we divide the range in half, and start with the middle value. If we meet the MSE allocated for that specific component, we take the upper part, and otherwise the lower part. In any case, we proceed in the same way until the range is lower down to one single value.

This method also allows to optimize each component separately, while ensuring the requirement of 37 dB is met.

V. Encoding the left image

Throughout the following discussion, we will assume that the left image is the reference image and that the right image is the target image. The following illustrates the encoding process for each of the components of the left image.

1) Preprocess the image by subtracting the value 127 to each of the pixels. This makes all the values to be centered around 0.
2) Pad the image with zeros to make the size of the image appropriate to perform a cascaded wavelet transform of 5 levels for component Y and 4 levels to components C_b and C_r.
3) Apply a 5-stage *D4* wavelet transform to component Y and 4-stages to components C_b and C_r.
4) Quantize the image as described in Section IV. The result of this operation is denoted as L_{Q_k}.
5) Encode L_{Q_k} via arithmetic coding.

The padding is done by adding four rows to the Y component and two to the other ones. We choose the *D4* wavelet transform after finding that it resulted in a lower entropy representation than with the *Haar* wavelet. Due to the fact that the Y component is four times bigger in size than the C_b and C_r components and after some simulations, we set $a_Y = 10/12$ and $a_{C_b} = a_{C_r} = 1/12$ for the quantizer.

VI. Encoding the right image

We have two modes to encode the right image. One of them encodes the right image independently of the left image, namely intra mode, and the other one makes use of the correlation between the two of them to improve the compression rate, denoted by intra mode. Choosing between the two methods is done after encoding the left image. Specifically, we compute the reconstructed version of L_Y, denoted by $\hat{L}_Y$, when the quantizer step Δ_Y is set to twenty. If the incurred MSE is bigger than $\frac{10}{12} \frac{MSE^*}{2}$ we choose the intra mode, and otherwise we use the inter mode.

A. Intra mode

In this case, the right image is encoded independently of the left image. We perform the same steps as for the left image, i.e., subtract the value 127 to center the values around zero, pad the image, apply a D4 wavelet transform of 5 and 4 stages for the luminance and chrominance components, respectively, apply a uniform quantizer with variable step, where a_Y, a_{C_b} and C_r take the same values as before, and finally encode with an arithmetic encoder.

B. Inter mode

In this case, we encode R_k using $\hat{L}_k$ as side-information, which is available at both the encoder and the decoder. The idea is to reduce the number of bits needed to express R by exploiting the similarity between R_k and $\hat{L}_k$.

For each of the right components Y, C_b and C_r, we do the following.

1) Partition each image into 30×30 blocks.
2) For each block, find the corresponding 30×30 block in the same component of $\hat{L}$ that minimizes the MSE, and store the corresponding shift vector $V = (d_x, d_y)$.
3) Compute the residual image, denoted by R_{RES_k}.

The reason reason we use a block size of 30×30 is the dimensions of the image are both integer multiples of 30, such that we do not need to pad the image with zeros. Also, based on the results shown in Fig. 3, we find that we do not incur much loss in the number of bits we need to encode. Ideally, for the motion compensation, we would like to allow the encoder to look in the entire image $\hat{L}_k$ in order to find the best match. However, this drastically increases the complexity

of the encoder, the running time and the alphabet of the vectors we then need to encode. We notice that the displacement is mostly in the x component rather than in the y component. Therefore, as a trade-off, we restrict the motion vectors to be in the range $d_x \in \{-64 : 2 : 64\}$ and $d_y \in \{-6 : 1 : 6\}$. The residual images for each of the components are computed as

$$R_{RES_k} = R_k - S(\hat{L}_k, \{V_k\}),$$

where $\{V\}$ denotes the set of shift vectors and $S(\cdot, \cdot)$ sets each block of the image to the one indicated by the shift vectors.

Notice that knowing the residual and the set of shift vectors is sufficient to reconstruct the corresponding image. Next we describe how we losslessly encode the shift vectors and the residuals.

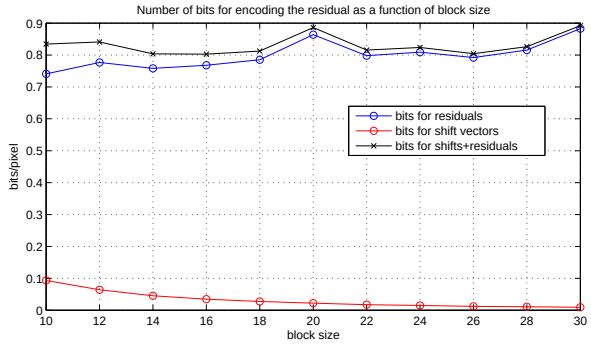

Fig. 3: Total bits versus block size

1) Shift vectors coding: We losslessly encode the set of shift vectors $\{V\} = \{(d_x, d_y)\}$ using a Huffman encoder, where each symbol is given by the pair (d_x, d_y). TABLE I shows the entropy of d_x, d_y and the joint (d_x, d_y) averaged over the set of images in the training set. Evidently, jointly encoding the shift vector components reduces the expected bit rate. There is a clear trade off between constructing the Huffman table based on the statistics of the set $\{V\}$ that we want to encode and constructing the table based on the statistics of the training set. We choose the second option due to time and complexity constraints, and to avoid the overhead caused by sending the Huffman table. The resulting distribution over the training set is shown in Figure 4.

$\mathbb{E}[H(d_x)]$	$\mathbb{E}[H(d_y)]$	$\mathbb{E}[H(d_x, d_y)]$
5.7007	3.8163	4.3926

TABLE I: Expected entropy for the shift vectors.

2) Residual coding: We found that after the quantization block with $a_Y = 10/12$ and $a_{C_b} = a_{C_r} = 1/12$, the residuals of the chrominance components were close to be zero for most of the images in the training set. Based on this fact, we impose the residuals of C_b and C_r to be zero, so that we do not need to send any bit of information for them. We then compute the MSE incurred by these two components as

$$MSE_C = \sum_{k \in \{C_b, C_r\}} \sum_{i=1}^{h_k} \sum_{j=1}^{w_k} (R_k - S(\hat{L}_k, \{V_k\}))_{i,j}^2$$

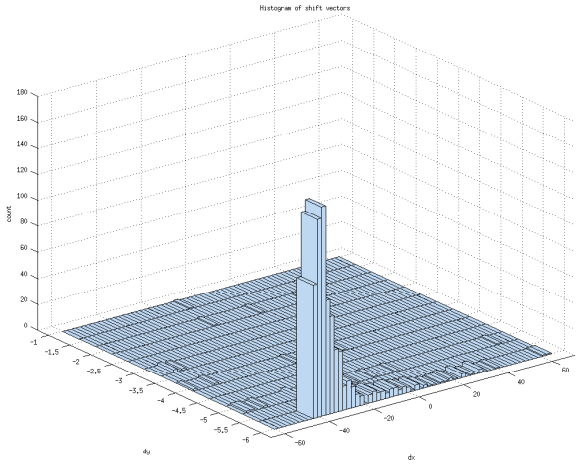

Fig. 4: Frequency of the shift vectors (d_x, d_y) for the training set

We apply a 30×30 *DCTII* to the residual of the Y component. We set the block size to 30×30 to be consistent with the block size used to find the shift vectors. We choose DCT over the discrete wavelet transform based on the approach in [3], and because of the edges between adjacent blocks due to the motion compensation.

After applying the *DCTII* transform, we quantize R_{RES_Y} with the uniform quantizer, with the parameter a_Y set to

$$a_Y = \frac{MSE^* - MSE_C}{MSE^*},$$

where MSE^* and MSE_C are as defined above. By doing so, we guarantee to meet the PSNR constraint.

After performing the quantizer step, we perform the following steps

1) ZigZag scan each block.
2) Substitute the last run of zeros, if it exists, by a new symbol.

In general the DC component tend to be high, and the AC components of the high frequencies to be close to zero, especially after quantization. By doing a ZigZag scan we expect to have a long run of zeros at the end. We make advantage of this by substituting this run by a new symbol and decreasing the size of the block to be encoded. Finally, we feed the resulting symbols to the arithmetic encoder. Notice that in this case, because we decrease the size of each block, the decoder also needs to know the length of the input to the arithmetic encoder.

C. Writing to file

We write to a binary file all the information needed by the decoder to reconstruct the compressed image. For each of the left components we allocate 8 bits for Δ, one bit for the sign of the minimum value after the quantizer, 15 bits for the absolute minimum value, 9 bits for the length of a binary sequence that specifies which values are found in the sequence that is fed to the quantizer, and the sequence itself. This information suffices to reconstruct the different values that can be found

at the input of the arithmetic encoder. Notice that the use of an uniform quantizer reduces overhead. We also need to store the output of the arithmetic encoder and the frequencies of each of the symbols. For the sequence, we allocate 21 bits to specify the length of the sequence, and then write the sequence itself. To store the frequencies, first we set the higher one to zero, and then we specify the number of bits we are going to use to describe each of the frequencies using 5 bits followed by the frequencies. Since the decoder knows the length of the input sequence to the arithmetic coder, it can reconstruct the frequencies.

When the right image is encoded using the intra-mode, we write to file exactly the same information as described above. For the inter-mode, we store the shift vectors encoded with the Huffman table. Since the Huffman table is stored in memory, we do not need to send it. For the residuals, since the chrominance components are set to zero, no information is needed to be sent for them. For the luminance component we send the same information as before regardless the output of the arithmetic encoder. The only difference is that due to the deletion of runs of zeros, the decoder no longer knows the length of the input sequence to the arithmetic coder, and therefore we need to allocate extra 19 bits to specify its length.

VII. Decoder

The decoder extracts all the necessary information from the binary file, and recovers the left and right images by just performing the reverse operations. As discussed below, a bottleneck in this step was the run time of the Huffman decoder.

VIII. Improving the running time

We found that performing certain computationally exhaustive tasks in C and C++ instead of MATLAB significantly reduced the running time. For example, using C++ to perform the block search reduced the running time of this operation from 87 seconds to 49 seconds. Also, using C to perform Huffman decoding on the Y component of the image reduced the running time needed for decoding from 9 seconds to 0.03 seconds.

IX. Simulation Results

TABLE II shows the performance of the proposed encoder for the set of 14 images in the training set. The results are shown in bits per pixel, computed as the size of the encoded file in bits divided by the size of the image, i.e. divided by 2×960. The first column specifies the picture being compressed, the second and third column show the bit rate achieved by the encoder when the right image is encoded using the inter mode or intra mode, respectively. The last column indicates the mode chosen by the encoder to compress the right image. As observed, the inter mode is not reducing the bit rate, contrary to what one would expect.

It is evident from the simulation results in TABLE II and the histogram shown in Fig. 4 that the choice of the d_y is causing our inter-frame coder to perform suboptimally. From the large frequency of d_y values near -6 in the histogram, it is clear that it is necessary to search through a greater range of vertical displacements. As discussed previously, the entropy values in TABLE I show that it makes the most sense to encode (d_x, d_y) jointly because of the reduced entropy (in bits per shift).

Image	Inter mode Bit rate [bits/pixel]	Intra mode Bit rate [bits/pixel]	Chosen mode
1	1.9236	1.7324	Intra
2	0.9556	0.7566	Inter
3	0.1807	0.1994	Inter
4	0.7164	0.6838	Inter
5	0.9304	0.8785	Inter
6	1.3679	1.239	Intra
7	1.9489	1.8104	Intra
8	1.8974	1.7188	Intra
9	0.6453	0.6391	Inter
10	1.998	1.7766	Intra
11	1.0631	0.9341	Intra
12	0.5879	0.5725	Inter
13	2.2172	2.1404	Intra
14	3.3196	2.3837	Intra

TABLE II: Compression results for the training set in bits/pixel for the two modes, and the actual mode chosen by the encoder.

X. Conclusions

We found that arithmetic coding outperforms Huffman encoding for our approach, and that using the DCT to encode the residual is advantageous over using wavelets because it results in a lower-entropy representation of the image.

A key conclusion from our approach is that is important to consider the trade-off between bits allocated to shift data and residual data. Our approach was based on optimizing the quantization step sizes for the reference and residual image. However, this approach depends heavily on the performance of the block search when computing the residual image. The quantization method that we used did not lend itself well to dynamic allocation of the blocks to neither inter nor intra coding. One possible extension to our project is to explore the concept of dynamically coding each block. A pitfall of our approach is that the suboptimal performance of the block search led us to encode the Cb and Cr components with zero bits.

In this project, we allocate half of the MSE to the left image and half to the right. This was motivated by our desire to achieve a pleasant looking image by equally allocating MSE to both images. We might have achieved a higher compression ratio if we removed this restriction.

In the early stages of this project, we experimented with the embedded zerotree wavelet (EZW) algorithm [4], which can lead to more improvements in compression. However, we found its implementation in Matlab to be far too long in running time. Thus, another possible extension of our project is to develop an efficient implementation of the EZW algorithm. Similarly, we anticipate that we can achieve further compression by combining bit planes with run length coding.

References

[1] M. Marcellin, M. Gormish, A. Bilgin, and M. Boliek, "An overview of jpeg-2000," *Image and Vision Computing*, 2003.

[2] J. Ellinas and M. Sangriotis, "Stereo image compression using wavelet coefficients morphology," *Image and Vision Computing*, 2003.
[3] T. Frajka and K. Zeger, "Residual image coding for stereo image compression," *IEEE Vehicular Technology Magazine*, January 2003.
[4] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, pp. 3445–3462, December 1993.

APPENDIX

CONTRIBUTIONS

Christopher Li

- Implemented block search and encoding of residuals in C++
- Zig-zag scanning code.
- Fast implementation of 2-d convolution via cconv in Matlab for use in wavelet transform.
- Implemented Huffman decoder in C.
- Experimented with embedded zerotree wavelets.
- Determined statistics and entropy of residuals and shift vectors.

Idoia Ochoa

- Wavelet transform and inverse wavelet transform.
- Optimization of the uniform quantizer.
- Comparison between the performance of the DCT and Wavelet transform for the residuals.
- DCTII transform and inverse DCTII transform.
- Substitutions of last run of zeros by a new symbol.
- Writing and reading from file.
- Huffman and Arithmetic encoder.
- Lossless encoding a Huffman table to write into a binary file.

Nima Soltani

- Implemented block search and encoding of residuals in Matlab.
- Speed up the search of the delta step for the uniform quantizer.
- Fast implementation of 2-d convolution via cconv in Matlab.
- Writing and reading to file.
- Developed initial implementation of residual coding with losslessly coded reference image.
- Explored performance under various block sizes.
- Code cleanup.