Stereo Image Compression

Deepa P. Sundar, Debabrata Sengupta, Divya Elayakumar
{deepaps, dsgupta, divyae}@stanford.edu
Electrical Engineering, Stanford University, CA.

Abstract— **In this report we describe in detail our algorithm for stereo image compression. The reference image was coded using a DCT, followed by entropy coding. The disparity and motion vectors for the other image are computed after a block-matching technique. Finally, the residual image is coded using a DCT, followed by entropy coding.**

Keywords- stereo image compression; block matching; DPCM

I. INTRODUCTION

With the increasing popularity of 3-D display of scenes using popular approaches as stereo or multi-view images, the need for stereo image compression has become more pronounced. In this competitive course project, our goal was to design and implement a stereo image coder and decoder which achieved minimum bit rate while meeting the specifications provided. The minimum allowed PSNR for any image was 37dB. There was also a time constraint of 2 minutes (on average) imposed on the entire coding and decoding process.

While existing standards like JPEG and JPEG2000 achieve significantly high compression ratios, there is still a lot of desired features for stereo image compression. This is because there is a great deal of redundancy between the left and the right stereo images, which can potentially lead to even better compression.

The rest of this report is structured as follows. In Part II, we discuss the process of encoding the reference image (left image). Various design choices that we made have been elaborated and justified. Part III discusses block-matching, prediction of the right image and residual computation. The decoder structures for both the left and right images is explained in Part IV. The performance of our algorithm is evaluated in Part V. Finally, Part VI summarizes our entire project and enlists some possible extensions to our work.

II. ENCODING OF THE LEFT IMAGE

A. Block based DCT

The reference image (which was the left image for our case) was encoded by using a DCT transform, followed by entropy coding [1][3]. While taking a block DCT, important design choices to be made were the block size to be used and the step size for quantizing the DCT coefficients. Considering the large size (540 x 960 pixels) of our training images, we felt that the standard 8x8 block would be too small and would increase both the bit rate and the computation time. As such, we decided on a block size of 32x32 for the Y channel of our images and a 16x16 block for the down-sampled Cb and Cr channels (by a factor of 2).

B. Quantization of DCT coefficients

We used a uniform quantizer for quantizing the DCT coefficients [2]. The step size of the quantizer determines both the bit rate and the PSNR of the reconstructed image. Since the problem statement required us to achieve a PSNR greater than 37dB while achieving minimum bit rate, we decided to independently optimize the PSNR of both the left and the right image to lie between 37dB and 38dB, since this would give us maximum compression. Starting with a step size of 20, we kept iterating until the PSNR of the left image falls within this range.

C. Differential encoding of DC coefficients

Since the magnitude of the quantized DC coefficients is quite large, transmitting them as such would require a large number of bits. Hence we decided to do a differential encoding on the DC coefficients of all the blocks in the image. The DC coefficient of the first block was transmitted as such, but for the remaining blocks, only the difference of each DC coefficient with the previous coefficient was transmitted.

D. Zig-Zag scan and Run Level encoding of AC coefficients

Following a DCT and quantization, a zig-zag scan was done on the quantized ac coefficients. Next, run-level coding was performed with the maximum length of run capped at 31. The trailing long run of all 0's was not transmitted. The 0-0 symbol (a 0 run of 0) was used as the EOB symbol.

E. Entropy Coding

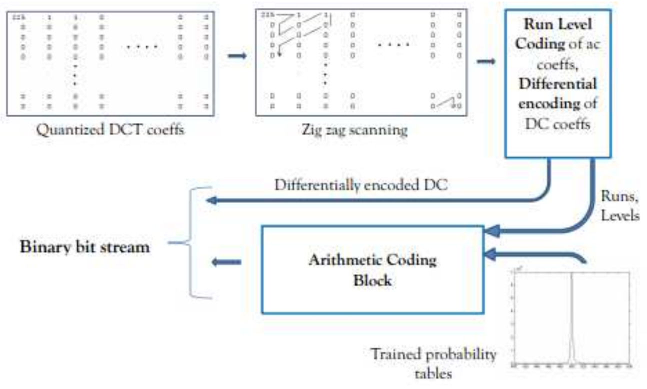
Following a run-level coding of ac coefficients, we decided to encode the runs and levels separately using a lossless entropy coder.

Initially, we implemented Huffman coder for encoding the runs and levels. Huffman tables were trained separately for every coefficient in the block. Thus, each coefficient in a block has a different Huffman table, while coefficient *i* in every block shares the same Huffman table. We encoded the DC and AC coefficients in every block using their respective Huffman table. The Huffman tables were transmitted along with the encoded bit stream to the decoder.

The main drawbacks with this earlier approach of ours were two-fold. Firstly, we were incurring a huge overhead in transmitting the Huffman tables. Secondly, even if the tables were hard-coded into the encoder and the decoder, we would not obtain a fractional bit rate (per pixel) since Huffman coding always uses at least 1 bit per symbol. These factors motivated us to implement arithmetic encoding for the reference image.

For using an arithmetic coder, we assumed that the alphabet for runs was {0,1,.., 31} while the alphabet for levels was {-200, -199,…, 199, 200}. The arithmetic coder needs the apriori probabilities of all the symbols in each alphabet in order to encode a given sequence of runs or levels. These probabilities were estimated by plotting appropriate histograms, which are shown in Figures (2) and (3). Finally, the runs and levels obtained from step *D* above were separately fed into the trained arithmetic coder, which generated binary bit streams which were then transmitted.

The entire process described in steps *A* through *E* is summarized in a block diagram shown in Figure (1).



Quantized DCT coeffs
Zig zag scanning
Run Level Coding of ac coeffs, Differential encoding of DC coeffs
Differentially encoded DC
Runs, Levels
Binary bit stream
Arithmetic Coding Block
Trained probability tables

Figure 1 : Block Diagram of Encoder for Left Image

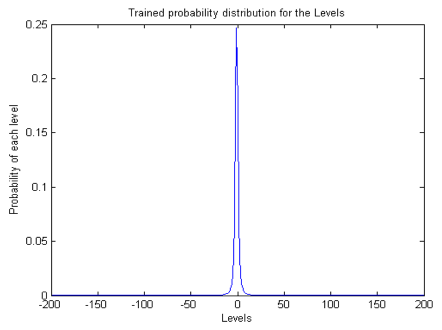


Figure 2 : Trained probability distribution for Levels

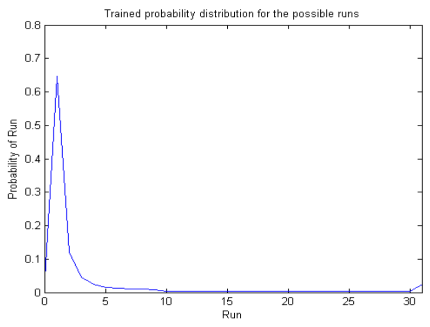


Figure 3 : Trained probability distribution for Runs

F. *Comparison with JPEG*

We first evaluated the performance of our algorithm for encoding the left image with standard JPEG compression as the base line. For a PSNR of approximately 37 dB, our algorithm out performs JPEG for a large number of test images. The results have been illustrated in the following scatter plot (Figure 4). From the plot, it can be observed that our performance is significantly better for certain images, while for other images, our performance is very close to that of JPEG standards.

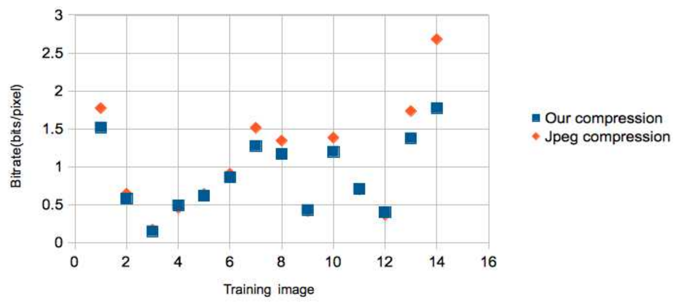


Figure 4 : Comparison of performance of our algorithm with JPEG (for encoding the left image)

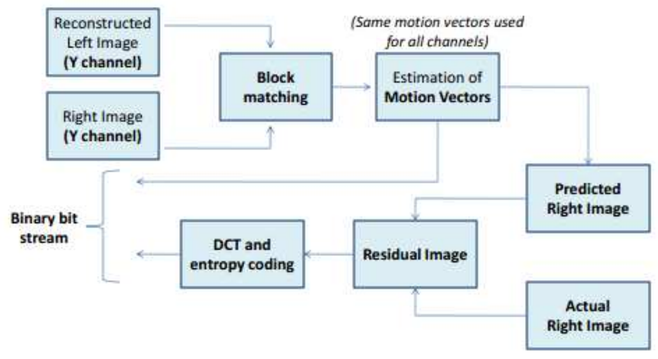


Figure 5 : Block diagram for encoding the right image

III. Block Matching and Encoding of the Right Image

In Stereo image compression, once the reference image is encoded independently, we do not have to encode the other image (in our case the right image) completely. We exploit the redundancy between the two images thereby reducing the bit rate. We essentially code the right image by computing a disparity map and the corresponding residual image. The residual image is then encoded in a similar manner like the reference image.

The block diagram in Figure 5 shows the encoding process of the right image. The RGB image is broken down into the Y, Cb and Cr channels and fed to the encoder. Once the left image is encoded completely, we use the optimized step size values for it to form the reconstructed left image. The reason for using the reconstructed left image is that, this is the image we essentially have at the decoder. Using the original left image to compute the motion vectors will not be what we actual do at the decoder and so would affect out PSNR optimization. Once the motion compensated right images are obtained, we then subtract these from the original right images to obtain the residual images. The residual images are encoded such that the PSNR of the right image is optimized to lie between 37 and 38dB.

A. Block Matching

Block matching is a technique in which for each block in the right image we try to find the best match in the reconstructed left image [4][7]. Here we exploit the fact that the color channels follow the luminance component. So we use the motion vectors computed based on the Y channel for the color channels – Cb and Cr (motion vectors are scaled by 2 in both x and y direction to account for down sampling). To find the best block match that would yield us the best predicted image we should ideally use full search algorithm. However full search is very slow for practical considerations. So we had to come up with some faster algorithm for block matching. In order to do so, we first had to estimate the motion range that would occur in the x and y direction. So we used 7 out of the 14 images provided for the training phase.

We chose a block size of 32 x 32 to perform the block matching. The choice of the block size was chosen such that we would want to minimize the bit rate spent on motion vectors and at the same time be able to capture all features in the image correctly, i.e. have as small a residual image as possible. Although block size of 8 x 8 is really good for prediction it is way too small, in reference to the size of the image we were encoding. We found that 32 x 32 was optimal for both.

B. Determining the search window size

Generally in stereo images we assume the motion disparity to be mostly concentrated in the x direction. However the training set had very large motion both in the x and the y direction. Here in the training phase, we perform an exhaustive search to compute the statistics of the x and y disparity/motion. We chose a range from -250 to +250 with a step size of 1 for the x direction and -150 to + 150 with a step size of 1 for the y direction. The below histogram plot for the y motion vector as an example is shown below (Figure 6). These statistics collected for 7 training images helped us fix the search window range.

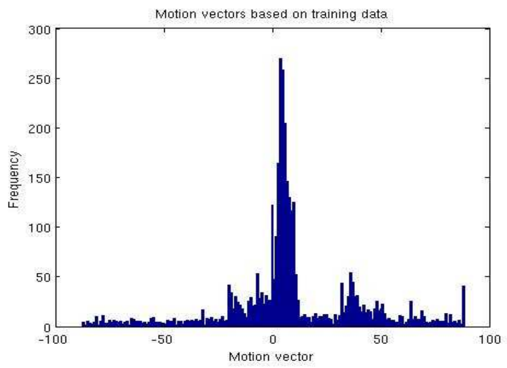


Figure 6 : Statistics for motion vector in the y direction (for 7 images in the training set)

C. Hierarchical block matching

Based on the search range we computed earlier we came up with a strategy to perform the block matching over a large range quickly and efficiently. We performed a hierarchical block matching with three levels of search. The first two levels are coarse search and the third level is a fine search algorithm. By doing a coarse – fine search we were able to search over a large range and at the same time perform faster. The metric used for the best match was SAD (Sum of absolute differences).

The block that minimizes the SAD would be chosen as the best match. We also choose a different step size for the x and y disparity as their search range varies.

The hierarchical block matching algorithm is as follows: For the current block in the right image, do the following search on the reconstructed left image.

Step 1: Search over -200 to +200 pixels with a step size of 12 for the x direction and -80 to + 80 with a step size of 5 for the y direction.

Step 2: Find the block that minimizes SAD.

Step 3: Once the best block is found, with that Index for x and y as the next level's zero motion vector, proceed to the second coarse search level.

Step 4: In the second level, search over the range of the first level's step size for the x and y direction. In this case, search over -12 to +12 pixels in the x direction and -10 to + 10 in the y direction. The step size for this level is halved. So for the x direction the step size is 6 and for the y direction the step size is 3.

Step 5: Find the block that minimizes the SAD at the second level.

Step 6: The third level is the fine search where the step size is 1 for both the x and y direction. Here the range is calculated as before. The x range in this level is from -6 to +6 pixels and for y it is -3 to +3 pixels.

Step 7: The block that minimizes the SAD in this level is the best match found.

The motion vectors thus obtained are then transmitted for both the x and y direction. These Y channel motion vectors are then scaled by a factor of 2 to predict the Cb and Cr right images. Here we choose a block size of 16 x 16 for the Cb and Cr channels so as to directly work with the derived motion vectors. A sample of the predicted right image is shown below in Figure 7.

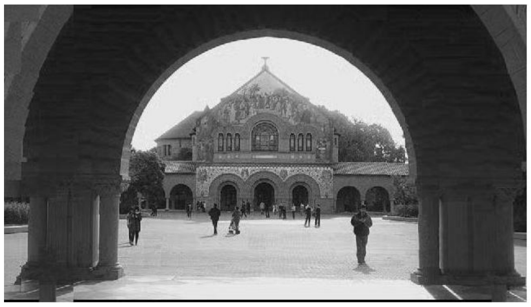


Figure 7 : Predicted Right Image

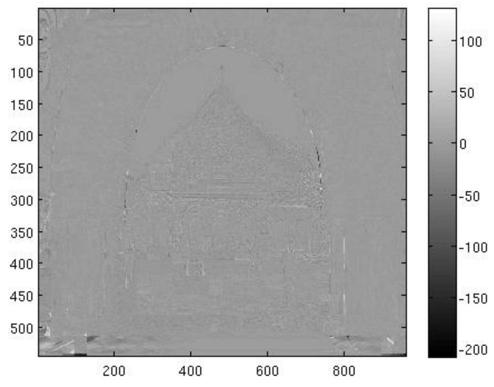


Figure 8 : Residual Image

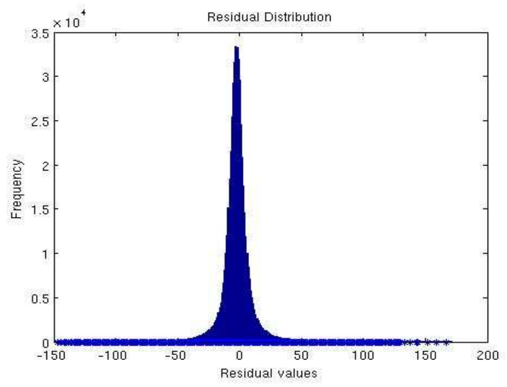


Figure 9 : Distribution of Residual values

The residual images obtained by subtracting the predicted image from the original right image are encoded in the same fashion as the reference image[5][6]. A sample of the residual image is shown in Figure 8. The residual distribution is shown in Figure 9. As expected we can see that the residual computed is small.

The motion vectors and the encoded residual bit stream are then transmitted to the decoder. The gain obtained by doing an entropy coding on the motion vectors versus directly transmitting the motion vectors to the decoder was way too small. So we decided not to encode the motion vectors for less computational complexity.

IV. Decoding of left and right images

In this part, we describe in detail the construction of decoder for the left and right images.

A. *Decoding the left image*

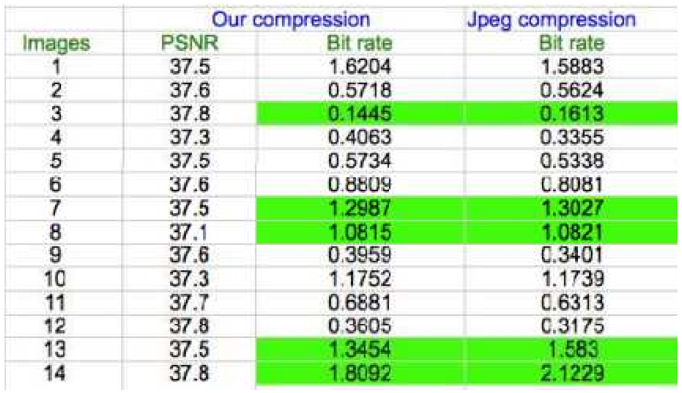
First, the dc coefficients are decoded. Each coefficient is decoded by simply taking the previous decoded coefficient and adding the difference (which was transmitted by the encoder) to it. For the ac coefficients, the binary bit streams for the runs and levels are separately fed in to the arithmetic decoder (which has the a-priori probability distributions of the various symbols hard-coded in it). It returns a stream of runs and a stream of levels, which are then fed into a run-level decoding block. This block reconstructs the quantized ac coefficients. Finally the ac and dc coefficients are fit together in an appropriate order to obtain the 32x32 block of quantized coefficients. The decoder then de-quantizes this by multiplying with the appropriate step size and takes an inverse DCT to get the reconstructed left image.

B. *Decoding the Right Image*

Each channel of the right image was first predicted from the reconstructed left image using the motion vectors that were transmitted. Then the residual for each channel was separately decoded using exactly the same process as described in section A above. Once the prediction and the residual was available, each channel of the right image was reconstructed by simply adding together the prediction and the residual.

V. Results

We compared the performance of our overall stereo image coding algorithm with JPEG as the baseline. The results of the comparison are tabulated in Figure 10 and illustrated through a scatter plot in Figure 11. It can be seen that the performance of our algorithm is comparable to JPEG and even beats JPEG for some images.



	Our compression		Jpeg compression
Images	PSNR	Bit rate	Bit rate
1	37.5	1.6204	1.5883
2	37.6	0.5718	0.5624
3	37.8	0.1445	0.1613
4	37.3	0.4063	0.3355
5	37.5	0.5734	0.5338
6	37.6	0.8809	0.8081
7	37.5	1.2987	1.3027
8	37.1	1.0815	1.0821
9	37.6	0.3959	0.3401
10	37.3	1.1752	1.1739
11	37.7	0.6881	0.6313
12	37.8	0.3605	0.3175
13	37.5	1.3454	1.583
14	37.8	1.8092	2.1229

Figure 10 : Results obtained on the provided 14 images

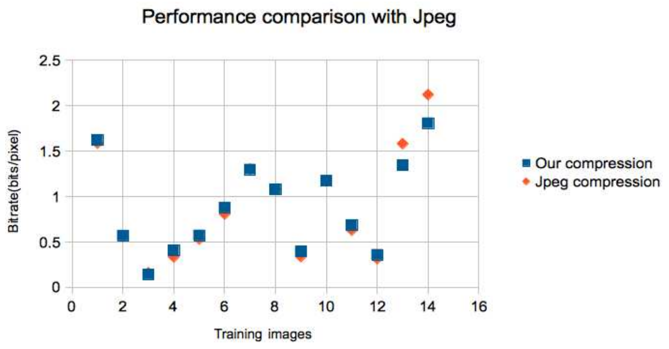


Figure 11 : Comparison of performance of our algorithm with JPEG (for both Left and Right Views)

For the test set, we achieved a file size of approximately 140kB (averaged over all seven images).

An example of decoded image from the training set is provided in Figure 12.

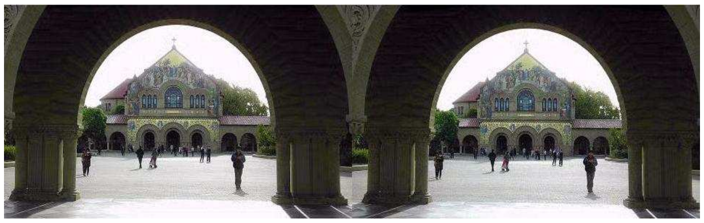


Figure 12 : Reconstructed Image

VI. CONCLUSION AND FUTURE WORK

Stereo image compression using DCT and entropy coding for the reference image , and disparity and residual encoding for the right image has been discussed in this report. From the analysis performed above, we can see that block DCT, Run Level coding, followed by Arithmetic coding performs quite well as compared to Huffman encoding. We successfully beat the JPEG standard with our algorithm for several images. In other cases, the performance was comparable.

Listed below are some other possible extensions of our work :

1. Intra mode selection can be employed for encoding the right image, when disparity prediction and encoding residuals does not give good performance.
2. Copy mode selection can be used for encoding the right image, when motion vector is zero and residual values are significantly small.
3. Sub-pel accuracy in block matching can be implemented to achieve a better motion compensated right image.
4. Variable block sizes in block matching to get better accuracy.

ACKNOWLEDGEMENT

We would like to thank Prof. Girod, Prof. Wiegand and the TAs Hari and Mina for providing us great feedback and guidance. This helped us a lot in refining our algorithm to perform better.

REFERENCES

[1] Mark S. Moellenhoff and Mark W. Maier, “DCT Transform Coding of Stereo Images for Multimedia Applications,” IEEE Transactions On Industrial Electronics, Vol. 45, No. 1, February 1998.

[2] A. Agarwal, “Compressing Stereo Images Using a Reference Image and the Exhaustive Block Matching Algorithm to Estimate Disparity between the Two Images “,International Journal of Advanced Science and Technology Vol. 32, July, 2011.

[3] M.Y.Nayan, E.A.Edirisinghe, H.E.Bez ,”Baseline JPEG-Like DWT CODEC for Disparity Compensated Residual Coding of Stereo Images” ,IEEE Proceedings of the 20th Eurographics UK Conference ,2002.

[4] T. Tao, J. Choon Koo, H. Ryeol Choi, "A Fast Block Matching Algorthim for Stereo Correspondence",CIS 2008.

[5] T.Frajka, K.Zeger, "Residual Image Coding for Stereo Image Compression",International Conference on Image Processing (ICIP)Rochester, New York, vol. 2, pp. 217-220, October 2002.

[6] Mark S. Moellenhoff, Mark W. Maier, "Transform Coding of Stereo Image Residuals",IEEE Transactions on Image Processing, Vol.7, No.6, June 1998.

[7] Won-Ho Kim, Jae-Young Ahn, Sung-Woong Ra, "An Efficient Disparity Estimation Algorithm for Stereoscopic Image Compression",IEEE Transactions on Consumer Electronics, Vol. 43, No. 2, May 1997.

APPENDIX

Distribution of Work

Deepa : Block Matching, Residual Computation, Training for Motion Estimation.

Divya : Block DCT and Quantization, Huffman Encoding, Simulation of test results

Debabrata : Differential encoding, Arithmetic coding and decoding, Training probability tables for arithmetic encoder and decoder

Report and Presentation had equal contribution from all team members