Efficient Compression Techniques for Stereoscopic Image Pairs

Charles Dunn, Oren Hazi, Marland Sitt
Department of Electrical Engineering
Stanford University
Stanford, CA
Email: {ccdunn, ohazi, msitt}@stanford.edu

***Abstract*—Recently, the amount of 3D media has been growing rapidly in the media industry. With its strong growth forecasted to continue, there is an increasing need to store and transmit 3D media efficiently. Conventional techniques fall short if applied directly, as they fail to take into account the high perspective correlation between the two views. In this paper, we examine several methods to code stereo image pairs by exploiting this statistical dependence. We demonstrate methods which outperform conventional techniques by a compression ratio of 25% on average over JPEG-2000 applied naïvely.**

I. Introduction

A. Background

A video sequence generally has many types of correlation. All sequences exhibit spatial correlation and temporal correlation. Spatial correlation characterizes the statistical dependence between different parts of the same image, while temporal correlation characterizes the correlation between frames at different time instances.

A 3D video image sequence is generally captured using a special camera with 2 lenses, spaced some distance apart horizontally [1]. This setup mimics the human visual system, and allows a 3D effect when the first sequence is presented to one eye and the second to the other. Due to the nature of the setup (Figure 1), we have a special kind of correlation, called binocular correlation or perspective correlation which is unique to stereoscopic image streams. The disparity between the two views for some point P in the field of view lies along the epipolar lines. When the optical axes are parallel, the epipolar axes will be parallel as well.

Currently, there are well established compression schemes which take full advantage of spatial and temporal correlation. These schemes are well suited for conventional image sequences, but do not adapt well when applied to a pair of 3D video image sequences in any direct way.

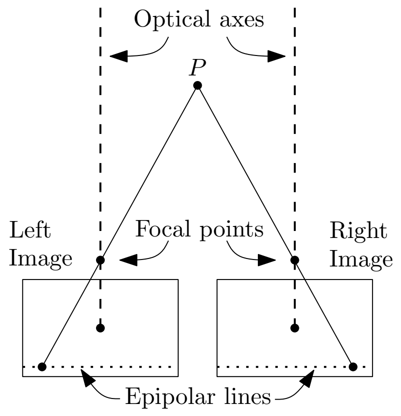

Fig. 1. 3D camera setup with parallel optical axes

B. Project Description

The goal of this project was to compress stereo image pairs as efficiently as possible. Given data included fourteen stereo image pairs in YCbCr color space with 4:2:0 chroma subsampling, at 8-bit precision.

An objective evaluation of the algorithm comes from a (peak signal to noise ratio) PSNR requirement. To be specific, each image must meet a minimum PSNR requirement of $\text{PSNR}_{\text{min}} = 37$ dB. Given the image passes this threshold, the algorithm will be judged on the number of bits used to encode each image.

The PSNR for an image is calculated in YCbCr space. We define the sum of squared distortion (SSD) between the original image component X_{orig} and the decoded image component X_{dec} as

$$\text{SSD}(X) = \sum_{x,y} (X_{\text{orig}}(x,y) - X_{\text{dec}}(x,y))^2. \qquad (1)$$

The SSD is calculated for each component Y, Cb, and Cr for both the left and right images. In particular, we calculate

$$\mathrm{SSD_L} = \mathrm{SSD(Y_L)} + \mathrm{SSD(Cb_L)} + \mathrm{SSD(Cr_L)} \tag{2}$$

$$\mathrm{SSD_R} = \mathrm{SSD(Y_R)} + \mathrm{SSD(Cb_R)} + \mathrm{SSD(Cr_R)} \tag{3}$$

Let the Y components have size h by w. Then, the mean squared error is given by

$$\mathrm{MSE_L} = \mathrm{SSD_L}/(hw + hw/4 + hw/4) \tag{4}$$

$$\mathrm{MSE_R} = \mathrm{SSD_R}/(hw + hw/4 + hw/4) \tag{5}$$

The overall MSE is the average of these two.

$$\mathrm{MSE} = (\mathrm{MSE_L} + \mathrm{MSE_R})/2 \tag{6}$$

The PSNR of the image is then

$$\mathrm{PSNR} = 10 \log_{10}(255^2/\mathrm{MSE}). \tag{7}$$

In addition to the 37 dB requirement, there was also a time constraint. The total runtime over 7 images is limited to 14 minutes on the SCIEN machines.

II. Methods

The overall strategy employed is to code one image conventionally using monocular image compression techniques. We term this the base image. Then the secondary image is reconstructed from the base image using disparity vectors. Specifically, the secondary image is divided into blocks of size $N \times N$, and a disparity vector is associated with each block which allows it to be reconstructed from the base image with small error. Residuals, which are the difference between the base image and the disparity compensated secondary image are also sent.

A. *Base Image Encoding*

In our attempt to exploit statistical dependence between the stereo images, we compress the left image using the proven methods of JPEG-2000 (JP2). Testing went into determining the ideal format of the data input to the JP2 algorithm. We compared three methods: separate, combined, and patched. Separate involved compressing the Y, Cb, and Cr components as three separate grey scale images, utilizing the fact that Cb and Cr have one quarter the data of the Y component in an attempt to increase base image compression ratio (CR). Combined involved upsampling the Cb and Cr components and converting to RGB format to input to JP2, hoping that the built in algorithms are best suited for a single RGB format. Patched involved compressing one large image composed of the Cb and Cr data added to the bottom of the Y data, hoping to take the advantages of encoding a single image without having to upsample Cb and Cr. The PSNR-CR plots are shown in Figure 2. A version of the combined method using the provided doubleres and halveres filters produced the highest CR while maintaining any PSNR. Our encoding scheme had to encode and then decode the compressed base image in order to calculate residuals that could be used to reconstruct the second image at the decoder.

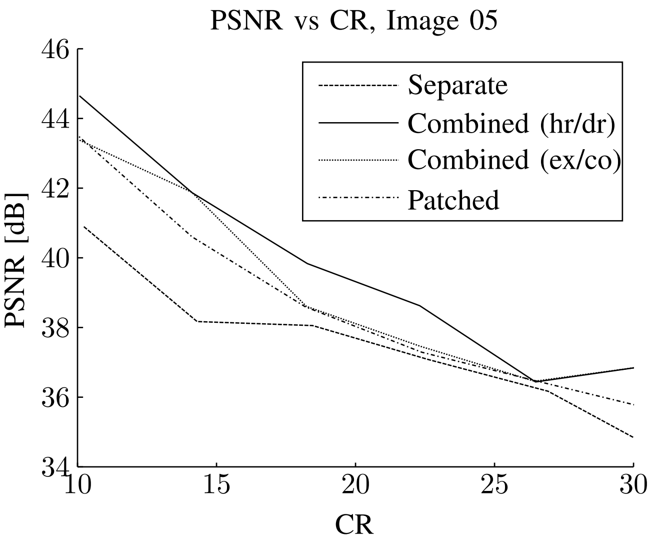

Fig. 2. Base Image Compression Comparison

Encoding the base image has a domino effect in that a base image PSNR well above the overall target PSNR will allow for more coarse quantization of residuals. However, a PSNR at the overall target PSNR will also evenly spread the quantization burden across the base image and residual compressions. Although we found that a base image PSNR about 1 dB above the overall target PSNR led to a higher overall CR, our algorithm counted on the advantages of equal PSNR goals for both base image and residual compression.

B. *Disparity Vector Estimation*

Disparity vector estimation is very computationally intensive and optimizations are necessary to avoid long runtime. A brute force method will search for the best disparity vector across the entire image. This kind of approach is infeasible, and one simple approach is to perform a search over a limited search window for each block.

Two separate methods of disparity vector estimation are proposed and implemented. The first we call top-down search and the second we call scale search. Both limit the search to rectangular windows around an initial guess of the actual disparity vector. Our initial guesses

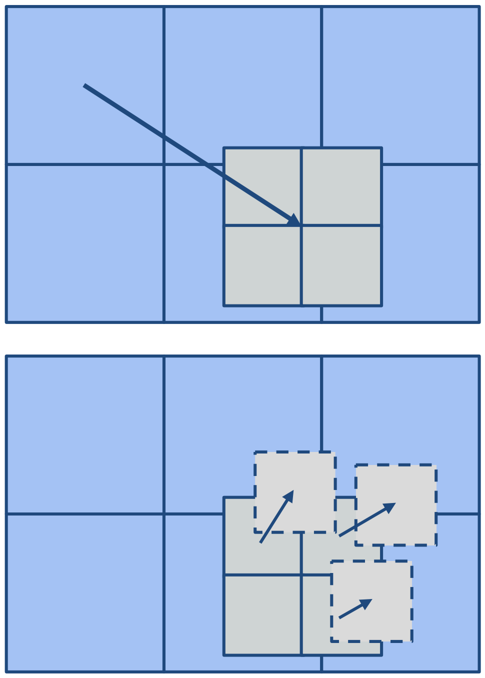

Fig. 3. Two iterations of the top-down search algorithm

of the x and y displacements were -35 and -8, respectively. This was an optimization specific to the image set we were given.

1) Top-down search: Top-down search is an iterative algorithm which successively narrows down disparity vectors for groups of blocks at a time. The assumption here is that disparity vectors vary slowly across blocks. If this is the case, we can perform an efficient search by lumping large numbers of blocks together. The first step of the algorithm finds the optimum disparity vector for extremely large block sizes with a very large search window, with an initial guess of the disparity vector. The second step will find disparity vectors for each subblock, using a smaller search window, centered at the optimal disparity vector location from the parent block. The top-down search algorithm is illustrated in Figure 3.

This algorithm enforces a smoothly varying disparity vector field for a given image since it searches a smaller window for each iteration.

We can trade off speed for accuracy by tuning the number of iterations and the sizes of the search windows at each iteration. The optimal initial guess for the disparity vectors come from analysis of the test set.

2) Scale search: The scale search algorithm attempts to find progressively finer approximations to the disparity vectors using scale pyramids of the left and right images. We begin at the smallest scale where we perform a search for block matches over a fixed window of integer shifts. This gives us a course approximation to the actual disparity vector values, since integer shifts at the small scale correspond to larger shifts at the native resolution.

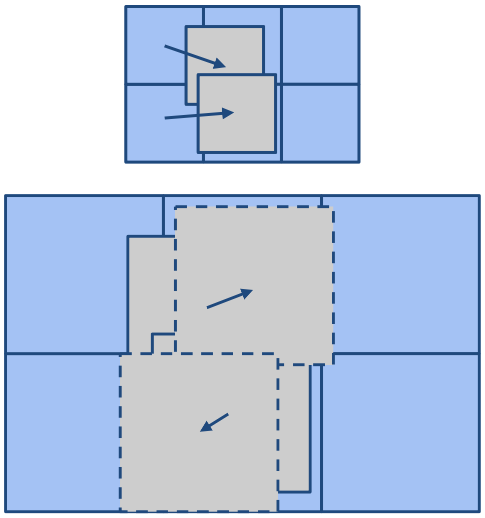

Fig. 4. Two iterations of the scale search algorithm

We then repeat the process using the next image in the pyramid, which is larger by a factor of two. We multiply the block dimensions and disparity vectors from the previous iteration by two. The search is then performed over the same window of integer shifts, but centered around the course disparity offset found in the previous iteration. This gives us a finer approximation of the disparity offsets, because our integer search window now corresponds to finer shifts at the native resolution. This process is repeated until we reach native resolution. This algorithm is illustrated in Figure 4.

This algorithm allows us to do a reasonably complete search of very large windows at native resolution. However, since it relies on a course to fine approximations, the algorithm can end up finding local optima rather than the true minimum SSD solution for a given window.

3) Quarter-pel disparity vectors: In addition to returning the minimum SSD disparity vector for integer block shifts, our windowed search function also returns a 5×5 grid of the SSD values surrounding the minimum SSD value. We use bi-cubic interpolation to fit a smooth curve through this grid, and then use the position of the minimum interpolated value to further refine the disparity vectors to quarter-pel precision.

C. Residual Quantization

After disparity vectors are calculated, we can form an estimate of the second image by taking the reconstructed

$$Q_{16} = \begin{bmatrix} 7&7&7&7&7&7&8&8&9&9&10&11&12&13&14&15\\ 7&7&7&7&7&8&8&9&9&10&11&12&13&14&15&17\\ 7&7&7&7&8&8&9&9&10&11&12&13&14&15&17&18\\ 7&7&7&8&8&9&9&10&11&12&13&14&15&17&18&20\\ 7&7&8&8&9&9&10&11&12&13&14&15&17&18&20&22\\ 7&8&8&9&9&10&11&12&13&14&15&17&18&20&22&24\\ 8&8&9&9&10&11&12&13&14&15&17&18&20&22&24&26\\ 8&9&9&10&11&12&13&14&15&17&18&20&22&24&26&28\\ 9&9&10&11&12&13&14&15&17&18&20&22&24&26&28&30\\ 9&10&11&12&13&14&15&17&18&20&22&24&26&28&30&33\\ 10&11&12&13&14&15&17&18&20&22&24&26&28&30&33&36\\ 11&12&13&14&15&17&18&20&22&24&26&28&30&33&36&39\\ 12&13&14&15&17&18&20&22&24&26&28&30&33&36&39&42\\ 13&14&15&17&18&20&22&24&26&28&30&33&36&39&42&45\\ 14&15&17&18&20&22&24&26&28&30&33&36&39&42&45&49\\ 15&17&18&20&22&24&26&28&30&33&36&39&42&45&49&52 \end{bmatrix} \tag{10}$$

base image and taking the disparity vectors into account. The error between this image and the actual second image are called the residuals. Sending these residuals is necessary in order to reproduce a good reconstruction. However, this is one area where we can save a lot of bits by sacrificing relatively little in image quality.

Quantization of the residuals is an important step in our algorithm. Residual quantization is the final step at which we apply any lossy methods. As our criteria was a target PSNR, it was important to get as close to the rate as possible on this step.

Given that the base image has a PSNR of PSNR$_{\text{base}}$, the second image must have a PSNR of at least

$$\text{PSNR}_{\text{target}} = -10\log_{10}(2 \times 10^{-37/10} - 10^{-\text{PSNR}_{\text{base}}/10}) \tag{8}$$

We transform the residuals using a discrete cosine transform (DCT) with block size $N = 16$. These residuals are then quantized.

We devised the following method to parameterize our residual quantization, so we could narrow down on the method which produced the PSNR closest to the target. Let

$$Q_8 = \begin{bmatrix} 8&8&8&9&10&10&11&12\\ 8&8&9&10&10&11&12&14\\ 8&9&10&10&11&12&14&15\\ 9&10&10&11&12&14&15&16\\ 10&10&11&12&14&15&16&18\\ 10&11&12&14&15&16&18&20\\ 11&12&14&15&16&18&20&22\\ 12&14&15&16&18&20&22&23 \end{bmatrix} \tag{9}$$

be the matrix to be used for 8×8 block sizes. Our 16×16 matrix is defined in equation (10). The matrix Q_{16} is meant for the Y component, and the matrix Q_8 is meant for the chroma components, which are downsampled by a factor of 2. It is worth noting that these matrices are constant along the antidiagonals. Since our images will be judged by a completely objective measurement (PSNR), there is no reason to take into account human perception.

Given a scale factor SF, we try the quantization matrices $A_{16}(SF) = Q_{16}/SF$ and $A_8(SF) = Q_8/SF$ rounded to the nearest integer. We perform a binary search over SF to narrow in on the optimal quantization level quickly.

As an additional optimization, we calculate the PSNR in the transform domain. For the second image, the difference between the original and reconstructed image is given exactly by the residuals. Further, since the DCT is an orthonormal transform, we can directly calculate the MSE in the transform domain.

D. *Disparity Vector Coding*

The geometry of the 3D camera configuration (Figure 1) suggests that we should expect small disparities for distant points in the scene and larger ones for nearby objects. In most image compositions, we would expect to see various objects at specific distances, as well as a backdrop like a wall or the sky at infinity. Since an object will generally consist of a contiguous region of pixels in the image, we expect to see many neighboring blocks with similar disparity offsets. To encode these disparities

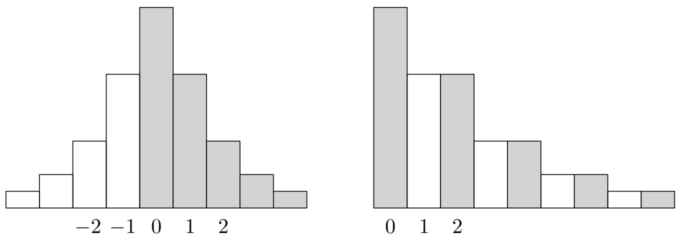

Fig. 5. Interleaving process for Golomb coding

efficiently, we decided to use differential coding.

The x and y components of the disparity vectors are coded separately. We first scan the matrix of disparity vector components using a zig-zag pattern, and then compute differentials for the resulting array. In our test set, these values resembled a two-sided geometric distribution centered around zero. We then map the negative values to positive values by interleaving (Figure 5), and encode the resulting stream with a Golomb coder. The encoder computes the optimal Golomb parameter to use based on the statistics of the particular array. This parameter is transmitted as the first byte in the coded stream.

The compression ratio for the encoded disparity vectors varied from 1.25 to 2.2 in our test set. The compression ratio for the y components tended to be higher by around 15%. This was most likely due to the smaller offset range in the y direction. Additionally, we expect large differences in the x direction whenever the image transitions from a foreground object to a background object and vice-versa. Such a change does not affect the y-component very much, so the differences tend to be closer to zero.

E. Residual Coding

Two separate methods of residual coding were implemented. The first uses Huffman coding on the DCT coefficients, and the second uses JP2 for the residuals. Both are described below.

1) Huffman Residual Coding: Once the optimal residual values are found, we must find a way of encoding them. Due to the fundamentally different nature of the DC and AC components, these are treated separately. We decided to use Huffman coding as opposed to the mathematically superior arithemetic coding due to the time constraint.

The DC coefficients are difference coded using Huffman coding, with the first coefficient transmitted as is.

The AC coefficients are run-level coded using Huffman coding. We go through blocks in a zig-zag pattern, which attempts to keep equal frequency components together. We also keep a separate symbol which signals the end of a block, and means the rest of the coefficients are zero. Joint encoding of the run-level pairs is the appropriate path to take when encoding the coefficients. However, we found the complexity of dealing with such a large Huffman table to be too high. Thus, we settled on encoding the run lengths and levels separately.

2) JP2 Residual Coding: Despite our hesitation to rely on predesigned compression algorithms, we elected to investige how JP2 compression would perform on our residuals. We knew that the statistics of the residuals were very different from a generic image for which JP2 was inevitably optimized. We still had a hunch that JP2 would outperform any intracoding algorithms implemented from scratch. JP2 is also very fast, so we saw no reason not to try JP2 residual compression in parallel with our own implementation of residual compression, followed by a selection of whichever method produced a higher final CR.

III. Results and Analysis

We tested our implementation on the provided 14 images. Although these same images were also used for training and parameter generation, we used the first 7 images more heavily for this, so in many respects the last 7 served well as unseen images for our code. Our preliminary results helped us select parameters and eliminate underperforming algorithms. Our final results, both in Huffman residual compression and JP2 residual compression formats, are provided and compared with naive JP2 compression.

A. Comparison of disparity vector search methods

Both our top-down and scale search algorithms performed reasonably well, but top-down search gave us better overall results. The disparity vectors found by either algorithm differed considerably, but we expect that this was largely due to configuration differences in search window size, as well as the subset of the search window was actually explored.

We also found that top-down search produced vector fields that varied more gradually, since the disparity vectors for specific blocks were close to the disparity vector of the parent macroblock. On the other hand, scale search was able to perform a search over a larger window, without dependencies between nearby blocks, resulting in more vectors with sharp discontinuities. These discontinuities would have been penalized by the Golomb coder.

TABLE I
COMPRESSION RATIO COMPARISON

Image	JP2	Huffman Res.	JP2 Res
1	10.0350	8.8253	10.0873
2	32.4873	36.6317	37.8367
3	195.3523	145.9870	154.1481
4	42.2930	52.0465	54.4423
5	26.4931	36.3025	37.8624
6	16.8792	19.8659	20.1422
7	9.7854	10.3284	10.6564
8	11.4483	14.6274	14.7235
9	44.2384	55.5488	59.1713
10	10.4480	13.5254	13.3902
11	25.5596	33.0044	34.9743
12	52.2791	61.9380	70.7295
13	7.9713	9.3048	9.3956
14	7.2194	6.2316	6.2801

B. Comparison of residual coding methods

Table I shows the final compression ratios obtained by our code when run on the 14 image training set. The JP2 column is compression ratios obtained by encoding the original images concatenated to one large image at a PSNR of 37 dB. The two right columns indicate the compression ratios obtained by our algorithm, whether encoding the residuals using Huffman encoding of their DCT coefficients, or simply compressing the residuals using JP2. In the limited training set, straight JP2 compression only beat our algorithm on two images. JPEG compression of the residuals obtained the highest compression ratio on nearly all the images, justifying our beliefs that the mutual information between the left and right images is significant, and that JP2 intra-image compression is extremely efficient. Nonetheless, our involved Huffman encoding of the residuals did outperform the other two methods on one image. We believe that images 1 and 14 were altered in some way, as they came with the stereo camera phone, and this certainly had an effect on how our algorithms performed in those cases.

C. Timing

Our final code was well below the 2 minute per image limit, though it did cause issues during algorithm devolopment as we often had to scale back or work around ideal algorithm implementation. A breakdown of our code's timing is shown in Table II. Over half the processing time is spent on the intensive disparity vector search. Our large search window demanded significant processing but also gave us great rewards in residual encoding. No other section of our code was notably time intensive in the final version, although interpolation and Huffman decoding both required special timing attention during development.

TABLE II
TYPICAL PROCESSING TIME PER IMAGE

Task	Processing time (s)
Base image encoding	3
Disparity vector search	36
Residual quantization	8
Residual Huffman coding	3
Golomb coding	1
Write to disk	1
Decoding	6
Total	58

IV. Additional Features

There is much room for additions to our code both in terms of the actual algorithm and the encoding time. A copy mode added to the cost function selection of disparity vectors could enhance our CR on large smooth areas of images. Variable block sizes would surely increase processing time significantly, but as the algorithm currently stands, we can afford to double the processing time while still meeting the original specifications. The disparity vector encoding could fairly easily be done as a joint distribution of x and y components since there was a high correlation between the two which we did not use to our advantage. We also believe there are many inefficiencies in our Huffman tables which could be implemented as binary trees and in our method of converting dictionaries to bit streams. In addition, a more scalable Huffman encoder and decoder would have allowed us to perform joint run-level pair encoding. Lastly, our method is extremely parallelizable, especially in the inner disparity vector search loop, and across our two residual encoding schemes.

V. Conclusion

We successfully implented an algorithm for lossy encoding of stereo images. We managed to outperform naive JPEG-2000 compression on nearly all of the images in the training set. When we did beat JPEG-2000, it was by an average additional compression factor

TABLE III
DETAILED WORKLOAD DISTRIBUTION

Task	Charles	Oren	Marland
Planning	34 %	33 %	33 %
Code			
General	35 %	20 %	45 %
Base image encoding	85 %	0 %	15 %
Disparity vector search	5 %	55 %	40 %
Residuals	60 %	30 %	10 %
Disparity vector encoding	0 %	100 %	0 %
Residual encoding	5 %	0 %	95 %
Testing	15 %	25 %	60 %
Presentation	55 %	20 %	25 %
Report	30 %	30 %	40 %

of 24.9% (or an additional overall CR of 6.97). Our Huffman residual encoding is very nearly comparable to JPEG-2000 residual compression, and even outperforms it occasionally. This means we relied only on JPEG-2000 for the intracoding of the base image, and all other compression was due to our own algorithm designs and implementations. Our algorithm met the timing requirements, impressively running in under one minute on average. It also meets the minimum PSNR, and just barely which was by design; our rate control produces compressed images with PSNRs usually less than 0.01 dB above the target PSNR. In short, we met all the system requirements, and all information suggests that our method is perfectly realizable as a real-world stereo image encoding scheme.

ACKNOWLEDGMENT

The authors would like to thank Professors Bernd Girod and Thomas Wiegand for their invaluable guidance and for teaching a truly stimulating course. In addition, we would like thank Mina Makar and Haricharan Lakshman for their help.

APPENDIX

A detailed workload distribution among the authors of this paper is given in Table III.

APPENDIX

The empirical distribution of our disparity vectors is shown in Figure 6.

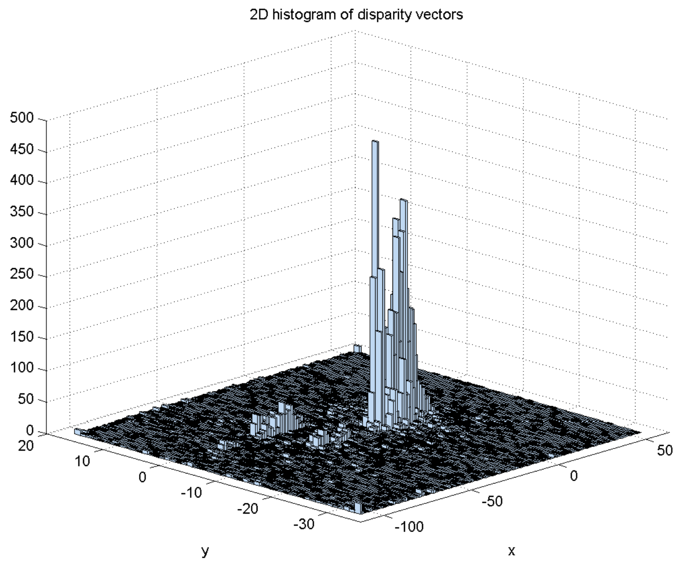

Fig. 6. Joint histogram of disparity vectors

REFERENCES

[1] T. Wiegand. (2012 Mar 14). “3D Video Coding.” [Online]. Available: http://www.stanford.edu/class/ee398a/.

[2] Rasmusson et al. “Frame Buffer Compression and Decompression Method for Graphics Rendering.” U.S. Patent 8031937, Oct. 4, 2011.