A Uniform Metric for Anaglyph Calculation

Zhe Zhang[a] and David F. McAllister[b]

[a]Program of Operations Research
[b]Department of Computer Science
North Carolina State University, Raleigh, NC 27695

ABSTRACT

We evaluate a new method for computing color anaglyphs based on uniform approximation in CIE color space. The method depends on the spectral distribution properties of the primaries of the monitor and the transmission functions of the filters in the viewing glasses. We will compare the result of this method with several other methods that have been proposed for computing anaglyphs. To compute the color at a given pixel in the anaglyph image requires solving a linear program. We exploit computational properties of the simplex algorithm to reduce computation time by 72 to 89 percent. After computing the color at a pixel, a depth-first search is performed to collect all neighboring pixels with similar color so that a simple matrix-vector multiplication can be applied. We also parallelize the algorithm and implement it on a cluster environment. We discuss the effects of different data dividing schemes.

Keywords: Anaglyph Stereo, Uniform Approximation, Linear Programming, Parallel Computing, CIE, Photoshop, Least Squares Approximation

1. UNIFORM ANAGLYPH CALCULATION

In this section we ill provide background for our method. In Section 2 we give a color comparison of different anaglyph calculation methods. In Section 3 we present how we accelerate the calculation in our method. We discuss the results of our acceleration techniques in Section 4. Conclusions and future work are presented in Section 5.

1.1. Introduction to Anaglyph Stereo Rendering

Anaglyphs have developed a bad name in the stereo community because of poor color representation and ghosting problems. The advantages of using anaglyphs vs. other methods of transmitting stereo are obvious and applications abound. We will not discuss them here.

Some popular algorithms for computing anaglyphs ignore the properties of the display device and the glasses. In Sanders and McAllister[1] we compared the Photoshop algorithm (PS), the Modified Photoshop algorithm (MPS), and the least squares algorithm (LS) proposed by Eric Dubois.[2] Our method is based on Dubois' approach, but we change the technique for measuring vector length and consider the computational algorithms necessary to compute the anaglyph in a reasonable time. As before, we ignore the important issue of retinal rivalry[3] which can create the appearance of ghosting. The conversion values given below are for LCD displays.

The conversion from RGB space to CIE space requires a linear transformation represented by the matrix C. The matrix C for LCD displays is:

$$C = \begin{pmatrix} X_R & X_G & X_B \\ Y_R & Y_G & Y_B \\ Z_R & Z_G & Z_B \end{pmatrix} = \begin{pmatrix} 0.4243 & 0.3105 & 0.1657 \\ 0.2492 & 0.6419 & 0.1089 \\ 0.0265 & 0.1225 & 0.8614 \end{pmatrix}$$

Further author information: (Send correspondence to Zhe Zhang)
[a]:Email: zzhang3@ncsu.edu, Telephone: 1 919 513 1907, [b]:Email: mcallist@ncsu.edu

The colors visible through a filter depend on the transmission function $f(\lambda)$ of the filter. The function f specifies the percentages of each visible wavelength λ transmitted by the filter. The product of the primary spectral distribution with the transmission function gives the spectral distribution of the primary as seen through the filter. The matrices Al for the left eye and Ar for the right eye convert the resulting filtered colors to CIE coordinates. For some common red/cyan glasses used below, these conversion matrices are:

$$Al = \begin{pmatrix} 0.1840 & 0.0179 & 0.0048 \\ 0.0876 & 0.0118 & 0.0018 \\ 0.0005 & 0.0012 & 0.0159 \end{pmatrix}, Ar = \begin{pmatrix} 0.0153 & 0.1092 & 0.1171 \\ 0.0176 & 0.3088 & 0.0777 \\ 0.0201 & 0.1016 & 0.6546 \end{pmatrix}$$

Eric Dubois[1] [2] showed how to use the transmission properties and spectral distributions for approximating colors in CIE color space to compute anaglyph colors. Dubois defined the length of a vector $\mathbf{x}$ = [X, Y, Z] in CIE space, using the Euclidean norm, $\|\mathbf{x}\|_2 = (X^2 + Y^2 + Z^2)^{\frac{1}{2}}$. The distance between two points x_1 and x_2 is therefore $\|x_1 - x_2\|_2$. In this case determining the closest vector in a subspace to a point not in the subspace is called least squares approximation (LS); the CIE color space is a linear space and approximation can be accomplished by a simple matrix multiplication for each pixel. LS approximation is equivalent to a projection in **R6** to the 3D subspace spanned by the 6 dimensional columns of the partitioned 6 x 3 matrix R defined below with right hand side partitioned vector d. The matrix C is the RGB-CIE conversion matrix defined above and v is the 6 component vector consisting of the left eye RGB color C_L and the right eye color C_R, $v = [C_L, C_R]^T$:

$$R = \begin{pmatrix} A_l \\ A_r \end{pmatrix}, d = \begin{pmatrix} C & 0 \\ 0 & C \end{pmatrix} v$$

The projection minimizes the Euclidean length of the vector $R[r, g, b]^T - d$.

This algorithm also uses scaling by a diagonal matrix N so that the white vector $w_3 = [1, 1, 1]$ in **R3** is mapped to the white/white vector $w_6 = [1, 1, 1, 1, 1, 1]$ in **R6**. The linear map can be written as $[r, g, b]^T = N(R^T R)^{-1} R^T d = Bv$. The B matrices for LS approximation is

$$B = \begin{pmatrix} 0.4154 & 0.4710 & 0.1669 & -0.0109 & -0.0364 & -0.0060 \\ -0.0458 & -0.0484 & -0.0257 & 0.3756 & 0.7333 & 0.0111 \\ -0.0547 & -0.0615 & 0.0128 & -0.0651 & -0.1287 & 1.2971 \end{pmatrix}$$

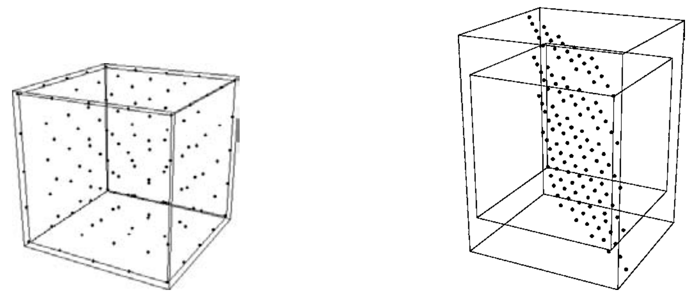
This algorithm was studied in Sanders and McAllister[1] and we omit the details. The B matrices for the PS and the MPS methods are also given there.

We note that the approximation may result in RGB components that are out of range. That is, some may be negative and some may exceed 1. Dubois recommends clipping to the RGB unit cube to solve this problem. Although this process involves a trivial computation, applying it to every pixel in an image adds to the time complexity. In addition, the method produces images that can be a bit dark. It does not preserve color equality as the Photoshop method does. That is, if $C_L = C_R$, it is not necessarily the case that the optimal solution is $[r, g, b]^T = C_L$. In Figure 1(a) we have chosen equally spaced colors in the RGB cube for $C_L = C_R$. Figure1(b) shows the optimal solutions in the RGB cube. Note how the solutions have been mapped to a parallelepiped and many lie outside the unit RGB cube.

Greyscale, however, is preserved in the sense that if v is a (nonnegative) scalar multiple of white, $v = \alpha W$, then the optimal solution is also a scalar multiple of white. This follows from the properties of norms and the fact that matrix multiplication is linear: $B(\alpha w_6) = \alpha B w_6$ and $\|Ax - \alpha y\| = \alpha \|A(\frac{1}{\alpha}x) - y\|, \alpha > 0$.

1.2. Uniform Anaglyph Calculation Algorithm

In this paper we change the length measure or the norm to the *Chebychev*, *minimax*, *infinity* or *uniform* norm of a vector $\mathbf{x}$ and define $\|x\|_\infty = \max\{|X|, |Y|, |Z|\}$. Approximation is in CIE space as in the Dubois calculation but we minimize the length of the vector $R[r, g, b]^T - d$ using the uniform norm (UN) instead.



(a) Equally spaced colors in RGB

(b) Images of points in part (a)

Figure 1.

As in the LS case, to ensure that the white vertex in **R3**, [1,1,1], is mapped to the white vertex in **R6**, [1,1,1,1,1,1], we use a normalization matrix N = Diag[6.60522, 3.23678, 0.00263908, 4.02254, 8.12129, 12.624]. To minimize the maximum deviation, we use a 7th variable ϵ and we formulate the approximation problem as a linear program (LP):

$$\begin{aligned} minimize \quad & \epsilon \; subject\ to\ the\ constraints\ (s.t.): \\ & |(R[r,g,b] - Nd)_i| \leq \epsilon, 1 \leq i \leq 6 \\ & r,g,b \leq 1 \end{aligned}$$

The Simplex algorithm computes only nonnegative solutions. Since ϵ is bounded below automatically, the problem becomes a 7-variable LP with 15 constraints (the absolute value constraints are converted to two constraints each).

We can write the constraints as follows:

$$\begin{aligned} minimize \quad & \epsilon \; subject\ to\ the\ constraints\ (s.t.): \\ & -\epsilon \leq R[r,g,b] - Nd)_i \leq \epsilon, 1 \leq i \leq 6 \\ & r,g,b \leq 1 \end{aligned}$$

where all variables are nonnegative.

As in the LS case it does not preserve color equality and does preserve greyscale.

1.3. Linear Programming

As stated above, equation 1 is a linear program (LP). An LP is an optimization problem where both the objective function and the constraint functions are linear, a canonical form being the following:

$$\begin{aligned} minimize \quad & c^T x \quad s.t. \\ & Ax \leq b \end{aligned} \tag{1}$$

A common method for solving an LP is the simplex algorithm, which is based on the observation that the feasible region of an LP, i.e., the set of all points **x** that satisfy the constraint functions $Ax \leq b$, is a polygon. The algorithm moves to adjacent vertices in a direction with a decreasing objective function value. An anti-circling scheme is used so that the algorithm will never consider a previously visited vertex. This scheme, combined with the fact that there are always finite number of vertices in the polygon, guarantees that the simplex algorithm finds the optimum in a finite number of steps.

While the simplex algorithm is conceptually elegant and normally efficient, the $O(2^n)$ worst case complexity can make it impractical for large scale problems. Since the 1980's algorithms with polynomial running time have been developed, the most important one of which is the interior point method. While the interior algorithm has a better asymptotic complexity, the simplex algorithm has less cost single step. In our problem the small size of the LP justifies using the simplex algorithm.

Solving an LP is considerably more computationally complex than applying a single matrix multiplication. Hence the problem produces interesting computational issues and in section 3 we seek parallel computation methods to accelerate the calculation to take advantage of coherence in digital images: the optimal solution at a given pixel is normally close to the optimal solutions of the surrounding pixels and the optimal solution for a pixel in a given frame of video is usually close to the optimal solution in the succeeding frame. Our goal is to produce anaglyphs of video signals at NTSC rates so we can apply the technique to problems like distance learning and virtual laboratories.

2. COLOR AND PERFORMANCE OF THE UNIFORM METHOD

2.1. Color Test

We compare the color, brightness and ghosting qualities of four algorithms: UN, LS, PS, and MPS. We created Gouraud shaded ellipsoids using 3D Studio. The hue and saturation over a given ellipsoid are the same for all pixels, but brightness varies from approximately 0 to 100 percent.

All ellipsoids are about the same size and the same distance from the viewer. We chose 64 RGB values that were equally spaced over the RGB cube, grouping them eight at a time. The colors are the coordinates of vertices of subcubes with edge lengths approximately $\frac{255}{3} \cong 85$. We consider all permutations of the intensities of 0, 85, 171 and 255 taken 3 at a time. The background is set to a neutral gray intensity of 166. There is no gamma correction. The cubes are numbered as shown in Table 1:

Cube number	R	G	B
Cube 1	0, 85	0, 85	0, 85
Cube 2	171, 255	0, 85	0, 85
Cube 3	171, 255	171, 255	0, 85
Cube 4	0, 85	171, 255	0, 85
Cube 5	0, 85	0, 85	171, 255
Cube 6	171, 255	0, 85	171, 255
Cube 7	0, 85	171, 255	171, 255
Cube 8	171, 255	171, 255	171, 255

Table 1. Cube numbering scheme

As an example, the images for cube 6 are shown in Figures 2 and 3.

2.2. Color Results

We did not attempt to perform a study in which the results were statistically sound since the effect of retinal rivalry cannot be predicted nor controlled. However, the comparisons for three non-colorblind observers were consistent, so we report them here. We compare each ellipsoid with the original left or right eye view *with glasses* (we note that comparing colors without the glasses is pointless and can result in misleading conclusions). We used three different pairs of red/cyan glasses: Reel3-D No. 7003, IMAX Fujitsu and the glasses provided by ABC to view the television show *Medium*. The results were independent of the glasses.

Our comparisons describe hue faithfulness except where noted. We also comment on brightness, ghosting[4] , and retinal rivalry[3] when the differences were significant. Since PS preserves color when the left and right eye values at a pixel are the same, we expect PS to be superior in hue and brightness approximation for most of the examples in our test. We comment below only when PS is not the best. Unfortunately, PS also inherits any problems with extreme retinal rivalry that exists when viewing the original left or right eye views through the

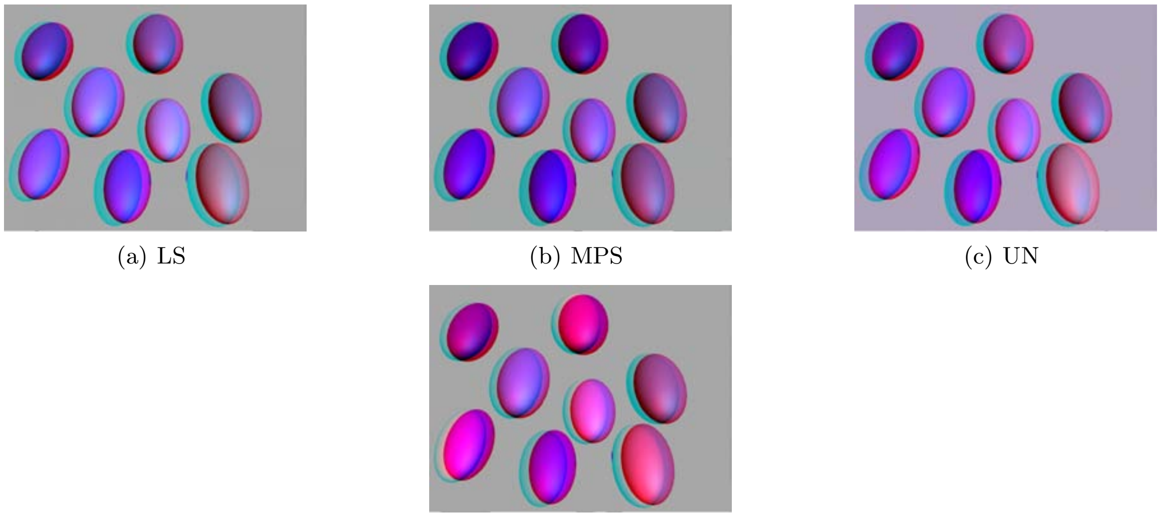

(a) LS (b) MPS (c) UN (d) PS

Figure 2. Cube 6 - Stereo image rendered with different methods

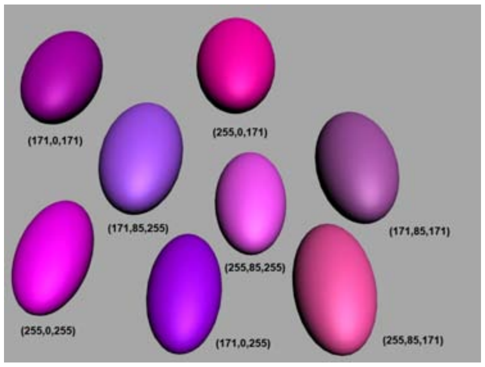


Figure 3. Cube 6 - Left eye view with true color

filters. This phenomenon is apparent in Cubes 2 and 5 described below. Hence, our comparisons are primarily between UN, LS and MPS. Recall that the red channel of MPS is computed by converting the left eye view to grayscale and inserting it in the red channel of the anaglyph. In regions of low green and blue intensities, the weighted average of R, G and B used to compute the intensity of the red channel may decrease the value of R in the anaglyph making the image darker. This phenomenon is evident in the results for Cube 2.

For cube 1, the point (0,0,0) or black, is the same for all methods, as was (0, 0, 85). UN is the best at points (85,0,0) and (85,0,85). PS is slightly worse, with MPS and LS equally bad. For (85,85,0), LS is only slightly better than MPS, and UN and PS are equally poor. LS is again the best for (0,85,0), then UN, then PS, and finally MPS. At (0,85,85) UN and LS are equally bad, and MPS is the worst. All are similar for the gray level (85,85,85) although UN is slightly darker.

For cube 2, shades of red, UN is best for all points. However, UN, while being a similar hue, is too bright, and there is conspicuous ghosting. LS and MPS are always too dark, but at points (171,0,0), (171, 85,85) and (255,0,0), LS is the worst, while MPS is worst at (255,0,85), (255,85,85), (171,0,85), and (255,85,0). MPS and LS are similar at (171,85,0). Retinal rivalry is a problem for PS.

Color cube 3 contains pale oranges and greens. The results are varied. At (171,171,0) UN is a similar hue, but too light. MPS is next in accuracy and LS worst. At (255,171,0), UN is again too bright, then LS, and MPS. UN, PS and MPS are equal, and LS is worst at (171,171,85) and (171,255,85). LS is best at (171,255,0), then

MPS, and UN last. LS and MPS are almost the same at (255,255,0), and UN is the worst. At (255,171,85), UN is the best, followed by LS, then MPS. UN is consistently too light, but the hues are closer to the originals than the hues of MPS or LS.

Cube 4 is shades of green. UN is the best at (0,171,0). PS is second, MPS is too light, and LS is too dark. For the points (0,255,0), (85,171,0),(85,255,0), (0,171,85), (0,255,85), (85,171,85), and (85, 255,85), there is a large amount of ghosting and retinal rivalry in PS. UN is consistently second best for these points, with a slightly different hue, but with the least ghosting and retinal rivalry. MPS is third with a different hue and too light. LS is worst with a much darker intensity and a different hue.

Cube 5 is the blue region. All ellipsoids have severe ghosting and retinal rivalry. At (0,0,171), UN is best, followed by PS and MPS equally. LS is too light and the wrong hue. UN is again the most accurate at (85,0,171), PS is too light, LS is the wrong hue, and MPS is too dark and the wrong hue. UN is the best approximation at (0,85,171) and (0,85,255). PS is slightly light, and MPS and LS are about the same-too light and the wrong hue. At (85,85,255), UN is the best, then MPS, LS, and PS. Extreme retinal rivalry makes color determination difficult for PS at (85,0,255). For this case, UN is second, then LS, and then MPS. PS is again the closest at (85,85,171), followed by MPS. UN and LS are equally too light and the wrong hue. The worst ghosting takes place at (0,0,255), with MPS best and the others almost impossible to see because of the retinal rivalry.

Cube 6 (Figures 3 and 2) contains the purple hues. UN is best for most cases. UN is best for (171,85,171) and (171,85,255), followed by PS which has more ghosting. LS and MPS were too dark, but MPS was the darkest. For the remaining points the order in decreasing hue approximation is UN, LS and MPS. For these remaining points there is significant retinal rivalry in PS. There is less retinal rivalry and ghosting is least for UN.

Cube 7 is the cyan region. There is considerable ghosting and retinal rivalry in PS. Second is MPS, with the right hue, but too light. LS is always the wrong hue, but the best brightness match. UN is consistently too dark and the wrong hue.

Cube 8 is the white corner of the RGB cube, the least saturated colors. All the methods produce approximately the same image for (171,171,171) and (255,255,255). All produce images of equal brightness at (171,171,255), but UN has a distinct rim in the ghosting area, an interesting phenomenon. At (171,255,171) LS is the wrong hue, MPS is worse, and UN is the worst but has the least amount of ghosting. UN is best at (255,171,171). LS is the wrong hue, and MPS is too dark and the wrong hue. At (255,255,171), MPS is best. LS is too bright, and UN is far too bright. There is considerable retinal rivalry in the PS approximation at (255,171,255), but it is the best hue approximation. UN and LS are about equal, and MPS is too dark. At (171,255,255) none of the three methods produces the correct hue. MPS is far too light, and UN is too dark.

No method is best for approximating all colors. The choice of a method depends on the colors involved, the importance of approximating brightness and the amount of ghosting produced. UN and PS were the most consistently accurate. PS is almost always better than MPS and requires the least calculation of all methods considered. LS usually produces a poor approximation for any hue. As mentioned above, retinal rivalry is severe in the cube 2 (reds) and 5 (blues) regions. We recommend avoiding these colors when possible. In most cases UN produces the least amount of ghosting and retinal rivalry without sacrificing brightness, a good reason to consider it in spite of the computational complexity discussed below. All the methods are good for unsaturated colors and for grayscale.

3. ACCELERATING THE CALCULATION

In the discussion below, the conclusions are based on the behavior of our methods for several continuous tone photographs. Our test examples can be found at our website at http://research.csc.ncsu.edu/stereographics/. Our computing environment for the uniprocessor case is a Dell 700m computer with 1.6 GHz CPU. Our parallel processing environment is an 8-node Cluster, where each node has dual 2.8-3.06 GHz processors. More specific information about the experimental environments and results can be found in section 4.

As stated in section 1.3, UN calculation is considerably more expensive computationally than the methods PS, MPS and LS, where only matrix multiplications are required. A stereo pair of 1085 by 675 images we test requires the unaccelerated UN method 127 seconds to render the anaglyph while the PS method requires less than 0.01 second. Our efforts to accelerate the UN calculation is described in the following sections.

3.1. Exploiting Color Coherence

There are many forms of coherence in computer graphics and image compression that are used to make rendering more efficient.[5] In our case the color of a pixel is often similar to that of its neighbors. This implies that the optimal solutions of the associated LPs should be "close" in the sense that solutions will often be the same or "adjacent" on the simplex defining the set of feasible solutions. We seek to find ways to exploit this in the simplex algorithm. More specifically, the linear programs for all pixels are similar in that they have the same objective function and constraint matrix, the only difference being the right hand values $\pm(Nd)_i$ of the constraints when put in the canonical form (1).

Moreover, according to Theorem 3.1 from D. Bertsimas and J. Tsitsiklis,[6] if the difference between each component of the right hand sides of two such problems is within a certain interval, and we have already computed the solution to one of the problems, then the other can be solved with a simple matrix-vector multiplication.

THEOREM 3.1. *Suppose that some component b_i of of the right hand side vector b is changed to $b_i + \delta$, and δ satisfies:*

$$\max_{\{j|\beta_{ji}>0\}}\left(-\frac{x_{M(j)}}{\beta_{ji}}\right) \leq \delta \leq \min_{\{j|\beta_{ji}<0\}}\left(-\frac{x_{M(j)}}{\beta_{ji}}\right)$$

($x_{M(j)}$ and β_{ji} are determined by the current optimal solution and the integer i). Then the optimal solution, as a function of δ, is given by $c_M^T M^{-1}(b + \delta e_i)$, where vector c_M^T and matrix M^{-1} are associated with the current optimal solution.

In our specific problem the difference between the right hand sides of two linear programs depend on the difference between the colors of the two corresponding pixels. Applying the theorem above, after computing the optimal solution for one pixel P of the image, we search the neighborhood of P to collect pixels that have colors that lie within the bounds given by the Theorem and for which we can use a matrix-vector multiplication to compute the optimal solution. Instead of solving a linear system to find x = M^{-1}y for each case, we precompute all ${}_{19}C_{15} = 3876$ matrix inverses for this problem (this includes slack and surplus variables) and ignore row permutation costs during processing. This reduces the complexity of finding x from $O(n^3)$ to $O(n^2)$ complexity, potentially a significant savings when processing video or large images. To test the efficiency of this concept we first design a simple search that investigates pixels in the same row of the pixel P for which we have an optimal solution. This simple search reduces the running time of the program by 55 to 65 percent on our uniprocessor system for the set of continuous tone images we chose.

A simple row search does not consider and exploit the speedup technique for all the qualifying pixels surrounding P. We then extend the procedure to include a depth-first search starting from pixel P. A stack technique is used where for each loop the head of the stack P is popped and its 8 *waiting* neighbors are examined. Those with color sufficiently close to P are pushed into the stack. The optimal solution for every pixel in the stack can be computed in accordance with Theorem 3.1. This continues until the stack is empty. This mechanism further reduces the running time of the program with simple search by about 45 percent for the set of images we chose.

In our implementation initially all pixels are marked as *waiting* or unprocessed. The program keeps a pointer to the next *waiting* pixel. We move the pointer, row by row, from the bottom to the top of the image. After computing an optimal solution at the current *waiting* pixel, we perform a depth-first search described above, process all qualified pixels with matrix-vector multiplication and mark all of them as *complete* or processed. Figure 4[7] illustrates how the depth-first search works. Figure 4(c) is the rendered stereo image, and Figures 4(a) and 4(b) are partially completed results of the program with the program pointer moved to fifth and half of the image, respectively. In Figure 4(a) and 4(b), pixels in gray areas are not processed yet, or in *waiting* status, and pixels in colored areas are processed, or in *complete* status. In Figures 4(a) and 4(b), we see how the algorithm progresses by exploiting the color coherence of these areas. In Figure 4(a), although the program *pointer* has only moved through one fifth of the image, the completed pixels include part of the sky and the man's cloth. In Figure 4(b) we can see most of the image has been processed, although the *pointer* has just moved through half of the image. The number of pixels processed is obviously larger than the number of LPs solved. In Figure 4(b), for example, at least all the colored pixels in the upper half of the image have been processed with the inexpensive matrix-vector multiplication, because the program pointer has never moved to that part.

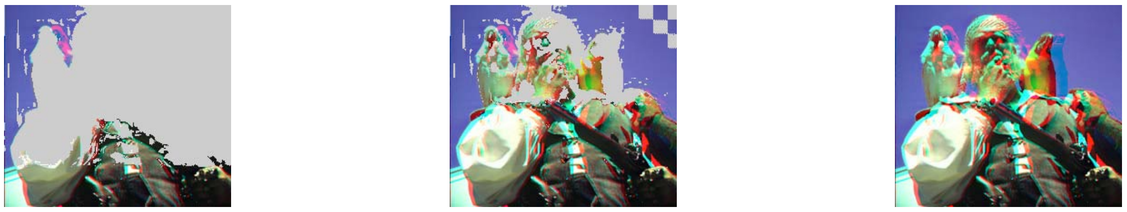

(a) Fifth-completed (b) Half-completed (c) The complete stereo image

Figure 4. Depth-first searching mechanism

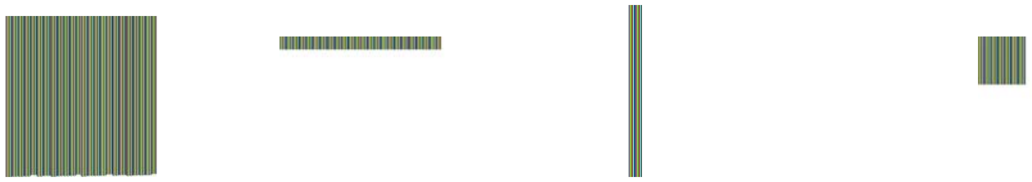

(a) The extreme example (b) Row-wise dividing (c) Col-wise dividing (d) Block-wise dividing

Figure 5. An extreme example

3.2. Parallel Processing

The increase in speed of the UN calculation described above is not sufficient to render images of size 1K by 1K at video rates on our uniprocessor system. Hence we sought to apply parallel processing techniques to further reduce the total running time.

There are obvious parallel computation techniques that can be applied to our problem that do not require an elaborate connection network and message passing. See Foley, van Dam, Feiner and Hughes[5] for an overview of parallel rasterization methods. A natural approach is to divide the image into several equal parts and pass each part to a different processor. If we do not include the depth-first search, computations for nonintersecting parts can be treated as completely independent; all pixels are processed independently.

However, if the searching mechanism is included, we expect that even for a given image, different data dividing schemes can result in considerably different performance. Here we show an extreme example. Consider the image in Figure 5(a) where all pixels in a column have the same color; for each column, the program has to execute the expensive LP calculation once. If we use the row-wise dividing scheme, each processor is assigned a set of consecutive rows of the image, as in Figure 5(b), and the color locality cannot be fully exploited. However, applying the column-wise method to divide the image, as in Figure 5(c), all the pixels in the column are processed at low cost. If we divide the image into nearly square blocks, each processor gets a block as shown in Figure 5(d), and we expect the execution time to lie between that of the other two methods (assuming, of course, processors of equal power).

Obviously, there is no static data dividing scheme that is best for all images if we exploit coherence within, but not among, blocks. We expect some processors to require more computation than others, because our algorithm takes advantage of color coherence which can be very irregular. To show this we applies our UN algorithm on several pairs of continuous tone images, with different numbers of processors and different data dividing schemes. As expected, we found that there is considerable difference between the amounts of running time of different computing nodes. We also notice a trend that the imbalance can grow with the number of processors applied to the problem. Since we are interested in processing video data, we seek a predistributed load balancing scheme that monitors processor computing requirements over successive frames and adjusts partitioning accordingly.

To study the problem we first develope a partition adjustment algorithm where workloads for different nodes on a static image are modified. We first run the parallel algorithm with static data dividing scheme, sort the execution times for each processor and then reallocate to attempt to attain the optimal allocation where each

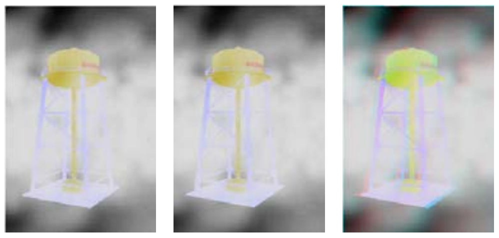

(a) Left eye view (b) Right eye view (c) Stereo image

Figure 6. A 203 × 305 pair of images

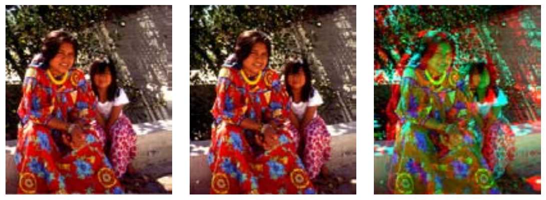

(a) Left eye view (b) Right eye view (c) Stereo image

Figure 7. A 230 × 261 pair of images

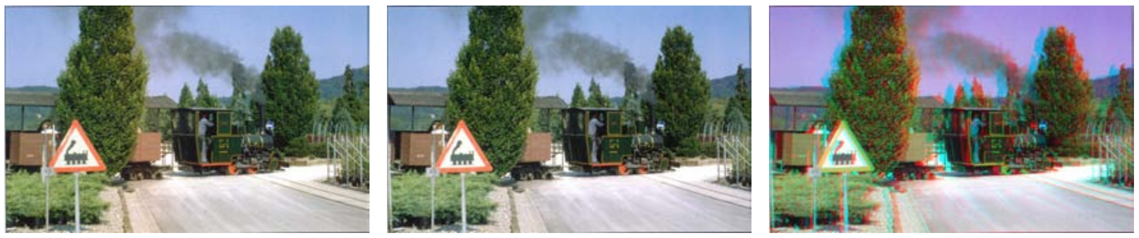

(a) Left eye view (b) Right eye view (c) Stereo image

Figure 8. A 1085 × 675 pair of images

processor requires the same amount of execution time(if such an allocation exists). The reallocation algorithm is discussed in section 4. Our experiments show that this can reduce processing time considerably. We are trying to extend this partitioning adjustment method for processing video.

Another focus of our future work will be to develop a method for processor communication so that workload can be adjusted during calculation on a single frame. In a message passing scheme there are many challenges, such as how to efficiently apply the depth-first search information across partition boundaries and how to reduce communication cost. In the following section we present our computation results.

4. ACCELERATION RESULTS

4.1. Results of Exploiting Color Coherence

We discuss the effects of exploiting color coherence. The experiments are performed on a Dell 700m laptop computer with Intel Pentium M 1.6GHz processor and 512MB memory. Three different implementations of the UN calculation are tested: the original program where **no search** is performed and colors of all pixels are computed using the simplex algorithm, the program with **simple search** along the row of the calculated pixel and the program with **depth-first search (D-F search)** around the calculated pixel, as described in section 3.1. To illustrate the effect of exploiting color coherence on different images we chose a set of image pairs with different sizes and features as shown in Figure 6 from Johnson and McAllister,[8] Figure 7 from Hannisian[9] and Figure 8 from our website.[7] The stereo image rendered with UN method is also displayed in the figures. The running times for processing these images are listed in Table 2.In all tables in this section and section 4.2 all times are in seconds.

	Size	**No Search**	**Simple Search**	**D-F Search**
Figure 6	203 × 305	11.02	2.51	1.19
Figure 7	230 × 261	9.98	4.82	2.80
Figure 8	1085 × 675	127.54	45.55	30.45

Table 2. The result of exploiting color coherence with different searching mechanisms

Simple search reduces the running time by 52 to 77 percent. The depth-first search further reduces the running time by 33 to 53 percent for a total speedup percentage of 72 to 89 percent.

For images with relatively complicated color patterns and strong color contrasts like Figure 7, the speedup from exploiting color coherence is not as pronounced as that for images with relatively simple patterns and weak color contrasts like Figure 6.

4.2. Results of Parallel Computing

In this section we will discuss the effect of applying parallel processing techniques to reduce the total running time. The experimental test bed is NCSU's IBM Blade Center Linux Cluster Henry2, with up to 16 computing nodes with 2.8-3.06 GHz Intel Xeon Processors and 4GB per node distributed memory. Our experiments are performed on the pairs of images in Figure 8 in section 4.1. In a parallel environment, running time is the time required by the longest running processor.

First we ran the original program (all pixels are processed with the simplex algorithm) on an increasing number of processors to determine how the running time and computing speed (reciprocal of running time) of the program vary with the number of processors. From Table 3 we can see that, as expected, there is an approximately linear speedup with the number of processors. Since there is no interprocessor communication or search, the speedup will be approximately linear regardless of the number of processors. The running time on one node is 103 seconds vs. the 127 seconds on the uniprocessor system because in this case, each node of the cluster has a much faster processor (2.8-3.6 GHz) than that of the uniprocessor we use in section 4.1 (1.6 GHz).

Number of nodes	1	2	4	8	16
Running time	103	56	27	16	10

Table 3. The running time varying with the number of computing nodes

Our next experiment was to run the program with a depth-first search on varying number of processors and different data dividing schemes. Row-wise, column-wise and block-wise data dividing schemes are applied, as described in section 3.2. Table 4 lists the result for each combination of number of processors and data dividing schemes. In the table, **Rows** $\times$ **Columns** defines how the image is divided among the computing nodes. For example, if the entry is 4×2, the image is divided by $8 = 4 \times 2$ processors into a 4 by 2 array. In the vertical dimension the image is divided into 4 equal parts and in the horizontal dimension it is divided into two.

Rows $\times$ *Columns*	Shortest running time	Longest running time
2×1	12.48	19.23
1×2	12.29	16.78
4×1	4.20	9.32
2×2	6.26	9.37
1×4	6.56	8.81
8×1	1.59	5.68
4×2	1.65	6.18
2×4	2.57	4.81
1×8	2.96	4.84

Table 4. Shortest and longest running times with different data dividing schemes

The imbalance among the workloads of different processors is obvious and it grows with the total number of processors. The largest relative difference occurs at 4×2, where the longest running time is 3.7 times the shortest. Also note the difference between the 4×2 and the 2×4 cases which we would hope to be approximately the same. It is also clear that imbalance exists for all the three dividing schemes described in section 3.2: row-wise, column-wise and block-wise. Considering the irregular patterns of images to process, we can further conclude that no static data dividing scheme will achieve balanced workloads among all processors.

We experiment with the static balancing algorithm introduced in section 3.2, where we sort running times of processors and repartition the workload to attempt to achieve uniform running times. We have tested the program on the images in Figure 8 with two processors, the 2×1 case above. First we partition the image equally where the lower and upper parts are assigned 337 and 338 rows of the images, respectively. In this case the running times are 19.23 seconds for the lower part and 12.48 seconds for the upper, a wide variation.

We then apply a simple algorithm that computes an adjustment according to the relative difference of the running times, i.e., taking half of the difference between the running times, computing its percentage in the total runtime and adjusting the number of rows accordingly. In this case the adjustment would be $\frac{19.23-12.48}{2} \times \frac{675}{19.23+12.48} = 60.4$, so the modified workload of the lower part should be 337 - 60 = 277 rows and the workload of the upper part should be 398 rows. However, as shown in the second part of Table 5 below, there is still imbalance between the running times of the two processors; because of the idiosyncrasies of this particular image the upper part has a longer running time, which means we have over adjusted the workloads. Thus, we try cutting the adjustment by half, i.e., from 60 rows to 30 rows. The result is shown in the third part of Table 5. We can see that although the upper part still has a longer running time, the imbalance is less. A similar approach can be formulated for more than 2 processors.

	Range of rows computed	Running time
Upper part	338 to 675	12.48
Lower part	0 to 337	19.23
Upper part	278 to 475	15.45
Lower part	0 to 277	13.14
Upper part	308 to 675	13.90
Lower part	0 to 307	13.52

Table 5. Results of different workload distributing schemes

By applying the static balancing approach we improve the performance from 19.23 seconds to 15.45 seconds in the first case, and to 13.90 seconds in the second case. Moreover, from the above table we can see the trend that if we keep modifying the adjustment according to the difference in running times, the dividing scheme will eventually converge to a balanced one where each processor has an approximately equal workload. This suggests that predistributed load balancing may be effective in processing video streams if we adjust the workload for a frame based on the processor running times for the previous frames.

4.3. Conclusion

In this section we list the results of several techniques to accelerate the UN anaglyph calculation. It is clear that our efforts were successful in reducing the time for rendering an anaglyph image using the UN algorithm. For the case of the 1085×675 image in Figure 3, the total processing time is reduce from 127 seconds (on 1 processor, before taking advantage of color coherence) to 4.84 seconds (on 8 processors, exploiting color coherence with depth-first search).

5. CONCLUSIONS AND FUTURE RESEARCH

We have described how to use UN approximation and the implied linear programming to solve an anaglyph computation problem. We compared the results with other methods to compute anaglyphs and concluded that there is still considerable work to be done.

Our future research will be focused on real time video rendering, with the challenge of exploiting the coherence between adjacent video frames and achieving a balanced workload among processors.

We also intend to examine approximation in uniform color spaces such as CIEL*a*b* and consider developing image processing filters to adjust images to reduce ghosting. Additionally, we are considering methods to modify algorithms so that solutions have the equal color preservation property.

ACKNOWLEDGMENTS

We wish to thank Sophia Sullivan for her work on creating the image pairs, comparing the color results and making comments to our paper. We also wish to thank Eric Dubois and his student Vu Tran for the LCD anaglyph data used in this paper. We acknowledge Xiaoming Hu for his help in designing the parallel program and analyzing the results.

REFERENCES

1. W. Sanders and D. F. McAllister, "Producing anaglyphs from synthetic images," *Proc. SPIE* **5006**, pp. 348–358, 2003.
2. E. Dubois, "Producing anaglyphs from synthetic images," *Proc. IEEE Int. Conf. Acoustics Speech Signal Processing* **3**, pp. 1661–1664, IEEE, (Salt Lake City, UT), 2001.
3. D. F. McAllister, *Stereo Computer Graphics and other True 3D Technologies*, Princeton U. Press, Princeton, NJ, 1993.
4. A. J. Woods and T. Rourke, "Ghosting in anaglyphic stereoscopic images," *Proc. SPIE* **5291**, 2004.
5. J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley Professional, 1995.
6. D. Bertsimas and J. N. Tsitsiklis, *Introduction to Linear Optimization*, Athena Scientific, Nashua, NH, 1997.
7. *Anaglyph stereo images*, NCSU Computer Science Department Research in Stereo Computer Graphics, http://research.csc.ncsu.edu/stereographics/.
8. T. Johnson and D. F. McAllister, "Realtime stereo imaging of gaseous phenomena," *Proc. SPIE* **5664**, pp. 92–103, 2005.
9. R. Hannisian, "Los huicholes," *stereo image on http://www.ray3d.com/road.html/* .