# Real-time stereo reconstruction through hierarchical DP and LULU filtering

François Singels, Willie Brink

Department of Mathematical Sciences
University of Stellenbosch, South Africa
fsingels@gmail.com, wbrink@sun.ac.za

## Abstract

In this paper we consider the correspondence problem for stereo-vision systems, where the aim is to reconstruct a scene in 3D from information captured by two spatially related cameras. Our main focus is to obtain dense reconstructions in real-time. We take a hierarchical approach to dynamic programming because of the good balance between accuracy and speed. We also apply a fast one-dimensional LULU filter at the various levels in the hierarchy for noise removal and scanline consistency. A sub-pixel refinement technique is also presented. Several experiments are reported on to illustrate the exceptional success of our approach at producing high quality results in near real-time.

## 1. Introduction

"Making computers see" — a problem that has been around for as long as the term *robot* has existed. One aspect of this problem is to estimate depth from the input provided by a number of cameras. Approaching it with the use of two cameras to generate a dense 3D reconstruction, which is termed *stereo-vision*, is akin to the way humans perceive the world. If we generate these reconstruction at near real-time rates it allows for some interesting applications, including:

- mobile robotics for obstacle avoidance, SLAM, navigation, etc.;
- body and pose tracking for human-computer-interaction;
- augmented reality where virtual objects are inserted in real world scenes;
- background subtraction/substitution.

The general problem in reconstructing a dense 3D model using stereo cameras can be summed up as follows: for each pixel in one image find a corresponding pixel in the other image. A combination of these correspondences and information of how the two cameras are related spatially facilitates the generation of 3D coordinates for each pixel, by triangulation.

When we say we aim to solve this problem in near real-time we define successful completion of this quest as being able to process each frame and produce results as fast as possible, but at least before the next set of input images become available. The frame rate at which the cameras operate restricts the amount of time available for calculations and, in return, the amount of time required determines the achievable frame rate.

The intricate part of a stereo-vision system, that of finding corresponding pixels between the two images, can be simplified by rectifying both images to remove radial distortion and transforming one image in such a way that each scanline (row of pixel intensities) in one image follows the same epipolar line as its corresponding scanline in the other image. This reduces the search for a corresponding pixel to one line.


Figure 1: An example of a dense 3D reconstruction from a stereo pair of images.

The distance between the coordinate of the reference pixel and that of the corresponding pixel in the other image is called its disparity. A map of disparities for every pixel in the reference image is the desired output of a stereo algorithm, and can easily be used to reconstruct a 3D model of the captured scene (assuming that the spatial relationship between the cameras is known).

Many different algorithms for calculating disparity maps have been attempted, each with its own advantages and disadvantages. Scharstein and Szeliski [1] provide a description and detailed evaluation of many techniques. Examples include:

- sum of squared differences (SSD);
- dynamic programming (DP);
- graph-cut optimization (GC).

The SSD algorithm is fast and highly parallelizable but finds correspondences very much on a local scale and is extremely prone to errors. The GC technique, on the other hand, solves a massive optimization problem and yields spectacular results but is computationally incredibly demanding, requiring typically minutes to process a single pair of images.

Since speed is of great importance in a real-time system we have chosen to adopt a hierarchical version of the DP algorithm [2]. We further compensate for the so-called scanline inconsistency problem (resulting from a lack of intra-scanline smoothness) by the implementation of a LULU filter.

# 2. Stereo-vision

Stereo-vision refers to the calculation of 3D points from the information in two spatially related cameras. This section provides an overview of the basic principles behind stereo-vision. We first provide a short description of epipolar geometry and how 3D coordinates can be calculated from a pair of corresponding points. We then explain image rectification and how it simplifies the search for corresponding points. Lastly we define and discuss the terms occlusion and disparity. A detailed description of stereo camera calibration, i.e. determining, among other quantities, the relative position and orientation of one camera to the other, is considered beyond the scope of this paper and the reader is referred to [3].

## 2.1. Epipolar geometry

Figure 2 depicts a diagram of a stereo. $C_1$ and $C_2$ are the camera centers in world coordinates, $L$ and $R$ are the image-planes of the left and right cameras, $x_1$ and $x_2$ are the corresponding points of the same feature in their respective image-planes and $X$ is the 3D position of that feature. If we assume camera matrices of the form

$$P_1 = K_1 R_1 \left[ I - \tilde{C}_1 \right] \quad \text{and} \quad P_2 = K_2 R_2 \left[ I - \tilde{C}_2 \right], \quad (1)$$

where $K_1$ and $K_2$ contain the intrinsic parameters of the two cameras, and $R_1$ and $R_2$ describe the rotation of each camera, then

$$x_1 = P_1 X \quad \text{and} \quad x_2 = P_2 X. \quad (2)$$

Since our aim is to reconstruct a 3D model from the images we need to work in the opposite direction from $x_1$ and $x_2$ in order to triangulate $X$. This triangulation is straightforward. The difficulty of stereo-vision is finding for a given point $x_1$ its corresponding point $x_2$.

For a dense reconstruction we need to find corresponding points for every pixel in one image. If $L$ is the reference image and $x_1$ and $x_2$ are measured in pixel coordinates, then for every pixel $x_1$ in $L$ we need to search through every possible pixel in $R$ to find a "best" match, which of course implies a horrendous amount of computation. Luckily the search space can be reduced drastically by the use of epipolar geometry.

We specifically use the fact that, according to epipolar geometry, the plane defined by the ray from $C_1$ through $x_1$ (and eventually through $X$) and the line from $C_1$ to $C_2$ cuts through $R$ in a single line, called an epipolar line. Finding $x_2$ is thus limited to a search along the epipolar line. To make the search even simpler we can make use of image rectification which is discussed next.

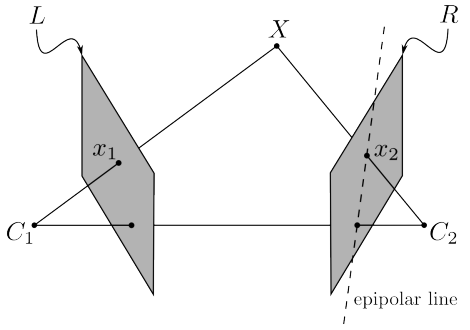


Figure 2: The epipolar geometry of two cameras.

## 2.2. Rectification

Image rectification builds upon the search constraints prescribed by epipolar geometry by attempting to projectively transform the images in such a way that the epipolar lines are perfectly parallel and horizontal. This would narrow the search for a correspondence down to a single scanline. Transforming such that the epipolar lines are parallel will require that the two image planes are coplanar. In an attempt to limit the distortion that a projective transformation causes, one should try to setup the cameras so that they are both more or less, and as closely as possible, facing the same direction perpendicular to the baseline (the line between $C_1$ and $C_2$).

Note that we need to rectify only one image, by mapping that image to the plane containing the other one. If the original camera matrices are $P_1$ and $P_2$ as defined before, and we wish to map the image produced by $P_2$, we define a transformation $T$ as

$$T = M_1 M_2^{-1}, \quad (3)$$

where $M_i$ denotes the $K_i R_i$ component of $P_i$ for each camera. Any point $x_2$ is now mapped according to

$$x_2' = T x_2, \quad (4)$$

such that $x_2'$ is the point on the rectified image. Figure 3 illustrates.

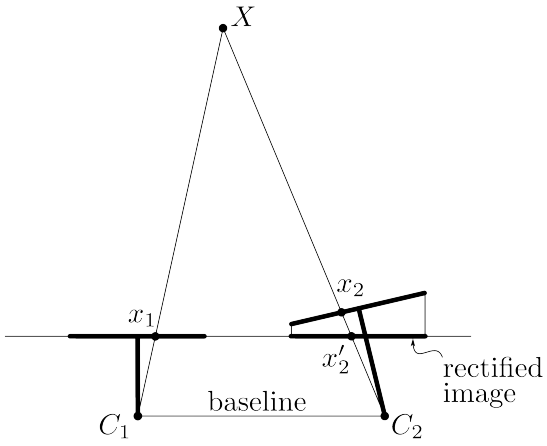


Figure 3: An illustration of the image rectification process.

The next issue to consider is possibility that there might not be a true match for every pixel in the reference image. A certain feature in the reference image could be obscured, or occluded, from the view of one of the cameras. This topic is covered in the next section.

# 3. Hierarchical DP for stereo

In this section we present some detail of the algorithm chosen for finding corresponding points in a pair of stereo images. First we provide some basic definitions of occlusions and disparities, and also give a few possibilities for dissimilarity measures. The normal dynamic programming (DP) algorithm for stereo-vision is then explained, after which we discuss how hierarchical DP is performed.

## 3.1. Definitions

The definitions discussed here are widely used in stereo-vision, and worth taking note of no matter what algorithm is used.

### 3.1.1. Occlusions and disparity

The phenomenon that occurs when some object or feature is visible in one image but not in the other is called an occlusion. Figure 4 gives an example of a view through a slit or gap. Occluded areas are marked in gray. Left-occlusion areas, i.e. areas not visible in the left camera view, are denoted by $L_{\text{occ}}$ and right-occlusion areas, i.e. areas not visible in the right camera view, are denoted by $R_{\text{occ}}$.

Figure 4 also illustrates what is meant by disparity. The pixel $x_2^*$ has the same coordinates as $x_2$, but in the second image. If $x_2'$ represents the same feature then the distance from $x_2^*$ to $x_2'$ is the disparity associated with $x_2$. Therefore the disparity of a pixel represents the distance that the pixel has moved from one image to the other. Intuitively, the larger the disparity the closer that feature is to the cameras.

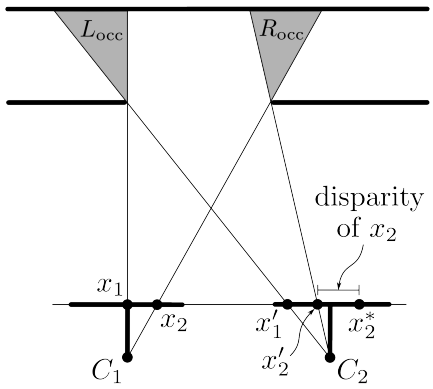


Figure 4: Examples of left- and right-occlusion areas, depicted in gray. The disparity associated with pixel $x_2$ is also marked as the distance from $x_2^*$ to $x_2'$.

### 3.1.2. Dissimilarity

For every pixel in a scanline of one image we should attempt to find a match in the same scanline of the other (rectified) image. In order to accomplish this some way of measuring the dissimilarity between two pixels is needed. The smaller such a dissimilarity, the more likely it is that the two pixels are a good match. There are various approaches of measuring dissimilarity ranging from simple but unreliable to reliable but computationally expensive.

A simple method would be to calculate the absolute difference between two pixels, as

$$D(L_{yx}, R_{y(x-d)}) = \left|L_{yx} - R_{y(x-d)}\right|, \qquad (5)$$

where $L_{yx}$ is the $x$th pixel in scanline $y$ of the reference image $L$ and $R_{y(x-d)}$ is the pixel in the other image it is being compared with (i.e. $d$ is the current candidate disparity offset). This method is extremely cheap but not robust.

The second possible method seeks to improve upon the previous one by summing over a small window around the pixels in $L$ and $R$, and then using the difference between the sums. Hence

$$D(L_{yx}, R_{y(x-d)}) = \frac{1}{n} \sum_{i=x-a}^{x+a} \sum_{j=y-b}^{y+b} \left|L_{ji} - R_{j(i-d)}\right|, \qquad (6)$$

where $n = (2a+1)(2b+1)$ is the number of pixels in the window. If we allow the center of the window to shift we can also compensate for edges in the image.

The method for measuring dissimilarity we prefer is one proposed by Birchfield and Tomasi [4]. It is insensitive to sampling because it considers the linearly interpolated intensities around pixels, as Figure 5 illustrates. Values $I_{R-}$ and $I_{R+}$ are first determined as the linearly interpolating values halfway between $R_y(x-d)$ and its two immediate neighbours. The minimum and maximum values of the set $\{I_{R-}, I_{R+}, R_{x-d}\}$ are then obtained, which we denote by $I_{\min}$ and $I_{\max}$. The function

$$D(L_{yx}, R_{y(x-d)}) = \max\{0, I_{\min} - L_{yx}, L_{yx} - I_{\max}\} \qquad (7)$$

determines the dissimilarity. Observe that if $L_x$ lies between $I_{min}$ and $I_{max}$ then $D(L_{yx}, R_{y(x-d)})$ is 0, otherwise $D(L_{yx}, R_{y(x-d)})$ equals the minimum distance between $L_x$ and the nearest boundary of the interval $[I_{\min}, I_{\max}]$.

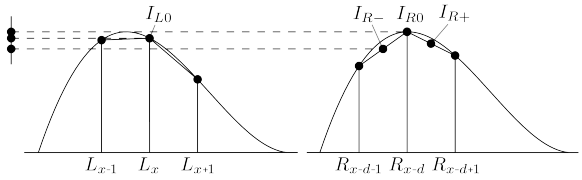


Figure 5: The dissimilarity measure we use, proposed by Birchfield and Tomasi [4]. See text for details.

### 3.1.3. Disparity space image

The disparity space image (DSI) is a matrix containing the dissimilarity values of two scanlines for every disparity in some range. Figure 6 shows two synthetic scanlines for illustration purposes. For the sake of simplicity in this example the absolute difference method given in (5) was used. For each disparity value $d$ the scanline $R$ is moved to the right by one pixel and then subtracted from $L$. The resulting DSI already gives us a hint as to where the matches should be. Correct matches are marked in gray. The diagonal arrows indicate right-occlusions and the vertical arrows indicate left-occlusions.

We have now covered all the necessary basic definitions, and can move onto the algorithm that finds optimal matches through the DSI.

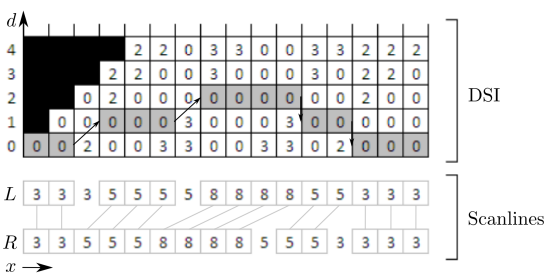


Figure 6: The disparity space image (DSI) for two synthetic scanlines. The true matches of the two scanlines are indicated in gray.

### 3.2. Dynamic programming

Dynamic programming is a process of solving a problem by dividing it into smaller problems recursively, solving the smallest problem first and using its answer to solve a slightly bigger problem, and so on. In stereo-vision the problem of finding the minimum cost path through the DSI can be solved in this manner. If we are able to calculate the minimum cost path up to a certain point in the DSI we can use that cost to calculate the cost of adding any point the path can move to next.

The possible moves a path can make in the DSI from a given point are shown in Figure 7. For the point $(d, x)$ in the DSI, i.e. at location $x$ in the scanline on disparity level $d$, the path could have originated from any white block and can go to any gray block next. By allowing the path to move in only one of these directions we enforce an ordering constraint. Horizontal moves incur a matching cost, since it implies that we are at the correct disparity and only the dissimilarity of the destination is added to the cost of the path. Diagonal and vertical moves incur an occlusion cost, since these moves imply that we are not at the correct disparity. In such a case the dissimilarity value carries little meaning. The occlusion cost is a user-specifiable value that should be expensive enough so that the path will rather follow a match. At the same time it should also be cheap enough so that it does not become too expensive to jump a few disparity levels through the occlusion to reach the correct matches.

We wish to calculate the minimum cost path for every pixel in a scanline, starting from the origin and working our way through the DSI left to right and top to bottom. After completion we can backtrack our way through the optimal path and simply map the disparities to each point on the scanline. At every point we compare the costs of the path coming from $(d+1, x)$, from $(d-1, x-1)$ and from $(d, x-1)$. In the first two of these we add the occlusion cost, and in the third we add the corresponding dissimilarity value. The lowest of these three costs is picked and saved as the cost to reach the current point. The coordinates of the chosen point, from which the path came, is also save for the purpose of backtracking.

The total cost of a path can now be defined as

$$C = \sum_{x=0}^{W-1} D(L_{yx}, R_{y(x-d)}|_{\text{match}}) + \sum_{x=0}^{W-1} (\beta|_{L_{\text{occ}}}) + \sum_{x=0}^{W-1} (\beta|_{R_{\text{occ}}}), \quad (8)$$

where $\beta$ is the occlusion cost. The first summation accounts for all the matches, the second for all the left-occlusions and the third summation accounts for all the right-occlusions.

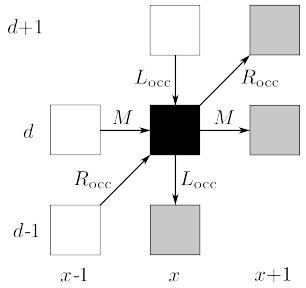


Figure 7: The possible moves that a path through the DSI can make. White indicates possible origins and gray possible destinations from the black block.

### 3.3. Hierarchical DP

Hierarchical DP, as proposed in [2], operates on very much the same principles as standard DP. However, instead of working on the originals, the images are first down-sampled several times. Down-sampling a 2D image is achieved by subdividing the image into groups of 4 pixels, and calculating the average of each. In the hierarchical approach standard DP is performed on the lowest sample level and those results are then used as an offset for the next minimum cost calculation, one sample level higher. This procedure is continued until the highest level, in other words the original image, is reached. The process yields a considerable improvement on the amount of computation required. It is no longer necessary to consider all possible disparity levels, except at the very lowest level. A small illustrative example follows.

If we consider the example scanlines from Figure 6,

$$L = [3,3,3,5,5,5,5,8,8,8,8,5,5,3,3,3]$$
$$R = [3,3,5,5,5,8,8,8,8,5,5,5,3,3,3,3]$$

and down-sample them to

$$L' = [3,4,5,6.5,8,6.5,4,3]$$
$$R' = [3,5,6.5,8,6.5,5,3,3],$$

we find the array of disparities for these scanlines is

$$D' = [0,\#,1,1,1,1,0,0].$$

Undefined disparities, indicated by $\#$ above, are given values by linear interpolation. We up-sample $D'$ by multiplying by 2 and doubling the width, obtaining

$$D_{\text{offset}} = [0,0,1,1,2,2,2,2,2,2,2,2,0,0,0,0].$$

These disparities can now be used as offsets in the original sample level. In order to find the new disparities we need only search in a small interval around each offset disparity. In this case we see that $D_{\text{offset}}$ is never more than one unit away from the correct disparities,

$$D = [0,0,0.5,1,1,1,1.5,2,2,2,2,1,1,0,0,0].$$

Notable criticism against the DP algorithm is that, although individual scanlines are matched well, there is no intra-scanline consistency. This comes as a result of scanlines in the reference image being matched independently of one another. The output typically contains severe disparity jumps in the vertical direction of the image, giving it a "streaky" appearance. The hierarchical approach does slightly better, due to the inherent image smoothing resulting from down-sampling. However, as soon as a scanline inconsistency is made on one level in the hierarchy, that error is propagated all the way to the top level. In the next section we discuss our addition to the algorithm that attempts to address this problem.

## 4. Improvements and post-processing

In this section we discuss how we use the LULU filter, pioneered by Rohwer [5], to improve upon the results obtained by the hierarchical DP algorithm discussed in the previous section. It should be mentioned that the LULU filter is not being applied only after the final disparity level has been calculated. Strictly speaking it is therefore not being used as a post-process, but since it is being applied *after* every level of disparity calculation is a post-processing operation of sorts. After the discussion of LULU we also discuss a possible sub-pixel refinement that can be performed on the final disparities.

### 4.1. LULU filter

Since in the DP algorithm disparities are calculated for each scanline independently, nothing is done to insure scanline consistency (or smoothness across scanlines). There might be radical differences among adjacent scanlines if, for example, the DP algorithm chose significantly different routes through textureless regions in the image where accurate disparities are hard to determine. In order to filter out these abrupt changes we apply the LULU filter.

The one-dimensional LULU filter applies a number of lower ($L$) and upper ($U$) sub-operators sequentially to a signal $x_i$ as follows:

$$L(x_i) = \max\{\min\{x_{i-1}, x_i\}, \min\{x_i, x_{i+1}\}\} \quad (9)$$

$$U(x_i) = \min\{\max\{x_{i-1}, x_i\}, \max\{x_i, x_{i+1}\}\}. \quad (10)$$

The $L$-operator removes positive outliers and then the $U$-operator removes negative outliers. In our case we want to use the LULU filter to enforce scanline consistency, so it is applied vertically across the scanlines. If a value is changed by any of the operators we check the preceding and succeeding disparities to ensure that the ordering constraint is not violated.

As mentioned before, an error or missed disparity in a low sample level can propagate through to the highest sample level and cause gross errors in the final result. For this reason we apply the LULU filter at every sample level, before it is upsampled, in an attempt to catch errors before they start propagating. Because this is a relatively cheap filter is has very little impact on the overall speed, while significantly improving the quality of the results. Some examples are given in section 5.

### 4.2. Sub-pixel refinement

In the discussion up to this point calculated disparities are obtained up to pixel resolution. The obtained 3D model will therefore, because of this discrete resolution, appear *planarized*. To smoothen it we need to find disparities to sub-pixel accuracy. Speed is still of great importance, hence we need a quick method of interpolation.

For each pixel $L_{yx}$ in the reference image we determine the dissimilarity between it and the three pixels $R_{y(x-1-d)}$, $R_{y(x-d)}$ and $R_{(y(x+1-d))}$ where $d$ is the disparity produced by the stereo algorithm. A quadratic polynomial is then fitted to these three dissimilarity values and its minimum, which may lie slightly to the left or right of $x$, is determined and taken as the new (continuous) disparity associated with pixel $L_{yx}$. Figure 8 gives an illustration. We refer to this procedure as sub-pixel refinement.

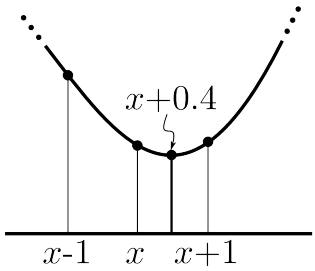


Figure 8: Sub-pixel refinement by determining the minimum of an interpolating quadratic polynomial.

## 5. Experimental results

For our experiments we used two *Point Grey FireFly MV* monochrome cameras each with a resolution of $512 \times 384$, synchronized through general IO pins at 15 frames per second. The software was executed on a standard desktop PC (*Intel Core2 Duo* 2.4 Ghz). Code was written in C++ using the *Code::Blocks* development environment, compiled with the GNU GCC compiler. *OpenCV* (open-source computer vision libraries) was used for calibration and rectification. The main program was split into several threads to make use of the multiple cores. The three most important threads are: the one that handles image acquisition, the one that calculates disparity maps (the most important) and the one that communicated with the graphics pipeline using *OpenGL* for model rendering. Test were performed indoors in an office environment where the lights could cause flickering if the shutter speed of the cameras were not long enough and textureless areas, such as the walls, could present potential problem areas for any stereo algorithm.

Figure 9 shows a typical image in (a) captured by the left camera. We used this image as the reference. The result of applying hierarchical DP without LULU filtering is shown in (b). Notice the gross error across the top of the head and the streakiness towards the sides caused by scanline inconsistencies. Introducing the LULU filter to the process yields the significantly improved result in (c). The image in (d) shows the lowest sampling level where the error originated. This error was successfully removed by the LULU filter in that sampling level, as can be seen in (e).

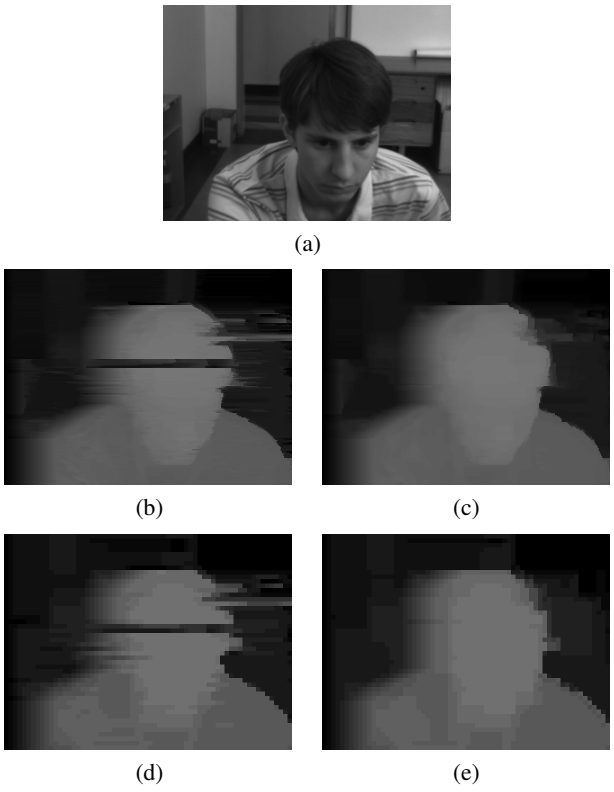


Figure 9: (a) Example reference image. (b) Result of hierarchical DP without LULU filtering. (c) Result with LULU. (d) Lower sampling level without LULU. (e) Same level as (d) with LULU.

Figure 10 illustrates the impressive effect that the sub-pixel refinement procedure has. A reference image is shown in (a) and the disparity map without refinement in (b) and with refinement in (c). Without refinement the 3D model shown in (d) suffers from large discontinuities between different disparities, and results in a *planarized* appearance. As shown in (e) important features such as the nose, mouth and chin become much more distinct in the presence of sub-pixel refinement.

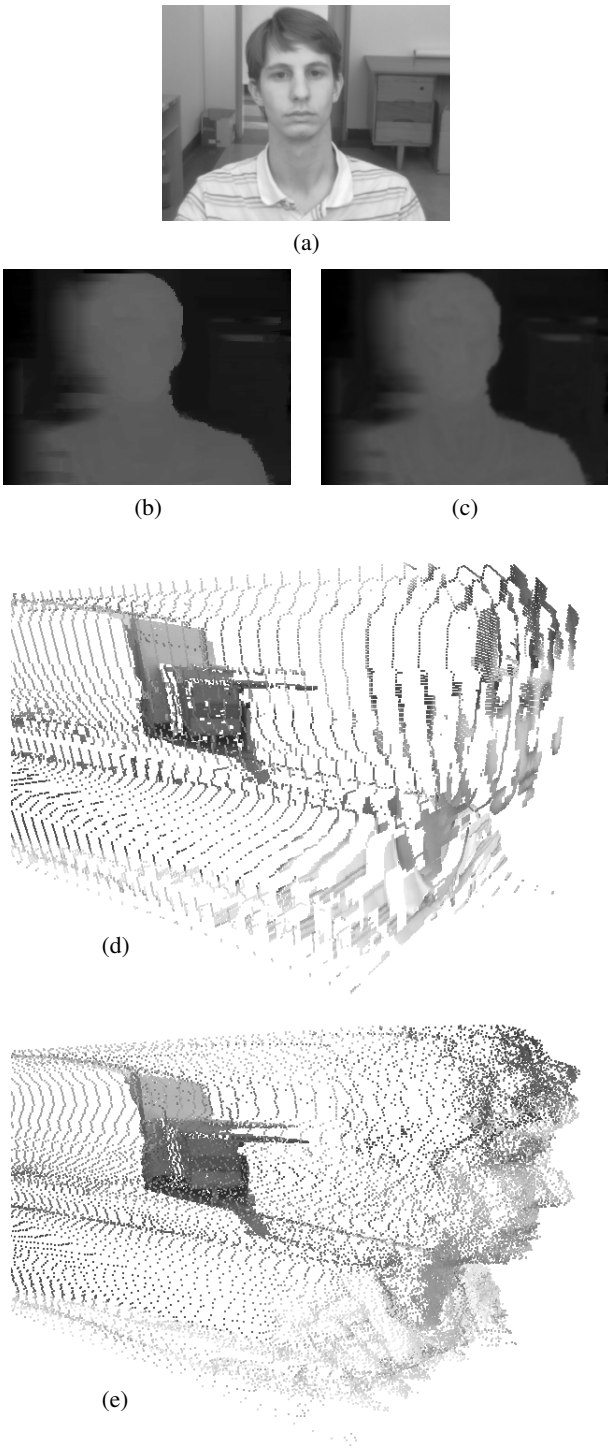

Figure 10: Effects of sub-pixel refinement on the 3D model.

The table below lists some execution times and potential frame rates achievable by our implementation. Initial SL refers to the processing of the lowest sampling level (the original image down-sampled three times) in the hierarchical DP (HDP), and SPR refers to sub-pixel refinement.

| description | time (ms) | FPS |
|---|---|---|
| initial SL | 9.49 | 105 |
| initial SL with LULU | 9.50 | 105 |
| HDP | 39.5 | 25 |
| HDP with LULU | 40.5 | 24 |
| HDP with LULU and SPR | 62.5 | 16 |

A final attribute of the HDP worth mentioning is that its execution time is not affected by the range of possible disparities, unlike standard DP. The result in Figure 10 contains large jumps in depth (between the background to the subject's face) requiring a large range of disparities to be calculated. The HDP managed successfully where standard DP would need vast amounts of processing.

## 6. Conclusions

In this paper we considered the correspondence problem for stereo-vision systems, with specific focus on obtaining dense reconstructions in real-time. We succeeded in this goal and implemented a system that currently runs at 15 frames per second and produces high quality results.

For the stereo algorithm a hierarchical approach to dynamic programming was taken as it provides a good balance between speed and accuracy. Further improvements were made by applying extremely fast LULU filters at every level in the hierarchy. A sub-pixel refinement technique was also discussed.

Experimental results are exceptionally satisfying. The LULU filter is not only efficient but also highly adept at catching and removing errors that cause scanline inconsistency. The sub-pixel refinement also yields pleasing results and removes the planarization found in most stereo algorithms that calculate disparities at pixel resolution.

Future work may include some post-processing on flagged occlusion areas and the investigation of various other techniques for "cleaning up" and possibly simplifying the 3D data. Some surface fitting algorithms to be applied to the output 3D point clouds is also a topic of interest.

## 7. References


[1] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms", International Journal of Computer Vision, 47:7–42, 2002.

[2] G. van Meerbergen, M. Vergauwen, M. Pollefeys and L. van Gool, "A hierarchical symmetric stereo algorithm using dynamic programming", International Journal of Computer Vision, 47:275–285, 2002.

[3] Z. Zhang, "A flexible new technique for camera calibration", IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(11):1330–1334, 2000.

[4] S. Birchfield and C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling", IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(4):401–406, 1998.

[5] C. H. Rohwer, "Variation reduction and LULU-smoothing", Quaestiones Mathematica, 25(2):163–176, 2002.