VIEW INTERPOLATION ALONG A CHAIN OF WEAKLY CALIBRATED CAMERAS

Dirk Farin[1], *Yannick Morvan*[1] *and Peter H. N. de With*[1,2]

[1] Univ. of Technol. Eindhoven, PO Box 513
5600 MB Eindhoven, Netherlands
d.s.farin@tue.nl

[2] LogicaCMG, PO Box 7089
5605 JB Eindhoven, Netherlands
P.H.N.de.With@tue.nl

ABSTRACT

This paper presents an algorithm for interpolating intermediate views along a chain of cameras. There is no restriction on the camera placement as long as the distance between successive cameras is not too large. The interpolated views lie on a virtual poly-line defined by the (ordered) set of cameras. Our algorithm requires no strong camera calibration as the necessary epipolar geometry is estimated from the input images itself. The algorithm first runs a preprocessing step to rectify the images to canonical epipolar geometry and to calculate disparity images. The actual view interpolation uses this data to synthesize intermediate views in real-time.

1. INTRODUCTION

Free-viewpoint video strives for displaying videos in which the user can freely choose the camera position. The two main techniques use either a geometric model of the scene, or the intermediate views are synthesized directly from the input data (image-based rendering). Like most practical approaches, we use a mixture of both to reduce the number of views required for the view synthesis and still be robust to the problems of geometry reconstruction. Our algorithm allows to synthesize intermediate views along a chain of cameras, where the cameras can be placed arbitrarily as long as the distance between successive cameras is not too large. This is the one-dimensional equivalent to the tri-view morphing [7] which allows the interpolation of views on the triangulated surface spanned by the set of camera positions. Our algorithm estimates the epipolar geometry between each pair of successive cameras and rectifies the input images to obtain horizontal epipolar lines (see Fig. 1). Disparity images are estimated for each pair of cameras and the intermediate views are interpolated in the rectified coordinate system. Prior to display, the interpolated views are projected back into the original, unrectified coordinate system.

2. EPIPOLAR GEOMETRY

Given a point $\mathbf{p}_a$ in one image, epipolar geometry tells us that the corresponding point $\mathbf{p}_b$ in a second image must be

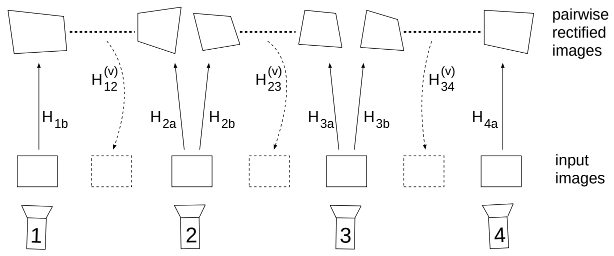


Fig. 1. The captured images are rectified pairwise. Disparity estimation and view interpolation between a pair of cameras is carried out on the rectified images.

located on a line $\mathbf{l}_b = \mathbf{F}\mathbf{p}_a$, and vice-versa, $\mathbf{l}_a^\top = \mathbf{p}_b^\top\mathbf{F}$, where $\mathbf{F}$ is a 3×3 rank-2 matrix, called the *Fundamental Matrix*. Hence, for each pair of corresponding points, $\mathbf{p}_b^\top\mathbf{F}\mathbf{p}_a = 0$ holds. The left and right epipoles can be extracted from $\mathbf{F}$ as the left and right null-spaces, respectively.

If the position of both cameras differs only by a shift along the horizontal axis, the epipolar lines are horizontal and corresponding epipolar lines have the same y value. In this case, the epipole is located at horizontal infinity. This situation is especially convenient for disparity estimation and view interpolation, since the parallax motion becomes a one-dimensional phenomenon in the horizontal direction. Unfortunately, because of manufacturing tolerances, it is very difficult to perfectly align a pair of cameras such that this case holds. However, for a pair of cameras, it is possible to virtually rotate the cameras such that we obtain horizontal epipolar lines. We then speak about rectified images. For more than two cameras, it is generally not possible to rectify the input images of all cameras at the same time. The best we can do is to rectify them pairwise.

2.1. Estimation of the Fundamental Matrix

We employ a combination of the 7-point and 8-point algorithm [2] to determine the fundamental matrix without the need for explicit calibration. For a static camera set-up, we also allow to use correspondences from multiple input frames, in order to obtain a more accurate estimate.

2.2. Image Rectification

We implemented the image rectification procedure proposed by Hartley [3]. This algorithms composes the rectification transformation from three parts. First, the coordinate-system origin is shifted to the image center by a translation $\mathbf{T}$. Second, the image is rotated ($\mathbf{R}$) such that the epipole lies on the x-axis. Finally, the epipole is mapped to infinity by the transformation

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1/e'_x & 0 & 1 \end{bmatrix}, \tag{1}$$

where e'_x is the Euclidean x-coordinate of the epipole after rotation. Thus, the final rectification transform is given by the concatenation $\mathbf{H} = \mathbf{G} \cdot \mathbf{R} \cdot \mathbf{T}$. The rotation matrix can be obtained easily from the normalized direction-vector to the epipole $\mathbf{d_e} = (e_x, e_y)^\top$ as

$$\mathbf{R} = \alpha \begin{bmatrix} e_x & e_y \\ -e_y & e_x \end{bmatrix}, \tag{2}$$

where $\alpha = 1$ if $e_x \geq 0$ and -1 otherwise. We have to distinguish on which side of the image the epipole is located, because otherwise the image could be rotated to an upside-down position. Even though this also represents a valid rectification, all disparity values would be inversed.

The obtained rectification transformation preserves the input image resolution at the coordinate-system origin. However, some parts of the image might be scaled down. This should be avoided since it decreases the quality of the interpolated views. Hence, we determine the factor m by which the image will be scaled down at a pixel (x', y') by

$$m(x', y') = \begin{vmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} \end{vmatrix} = \frac{1}{D^2} \left[\begin{vmatrix} h_{00} & h_{01} \\ h_{10} & h_{11} \end{vmatrix} - \begin{vmatrix} h_{00} & h_{01} \\ h_{20} & h_{21} \end{vmatrix} y' + \begin{vmatrix} h_{10} & h_{11} \\ h_{20} & h_{21} \end{vmatrix} x' \right], \tag{3}$$

with $D = h_{20}x + h_{21}y + h_{22}$. In non-degenerate cases, m_l is monotonic in x', y' and the minimum value $m' = \min(m)$ over the image area can be found in one of the image corners. Increasing the resolution of the rectified image by a factor of $\sqrt{m'}$ compensates for the loss of resolution.

3. DISPARITY ESTIMATION

The disparity image is estimated with a 1D dynamic programming algorithm, which is a slightly modified implementation of [1]. A commonly known problem of disparity-estimation algorithms based on dynamic programming is that the independent processing of scan lines leads to inconsistencies between scan lines, showing as horizontal stripes in the disparity image (Fig. 2(a)). In [1], this problem was

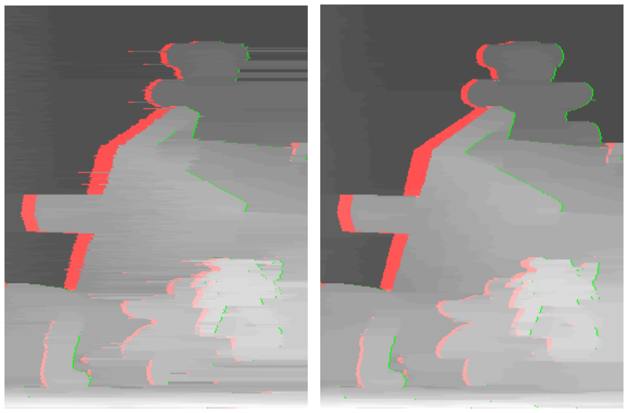

(a) no inter-line cost (b) with inter-line cost

Fig. 2. Estimated disparity image with (a) dynamic programming on independent scan lines, and (b) adding the difference to the previous scan line into the cost function.

addressed by classifying the disparity estimates into different levels of reliability and subsequent postprocessing of the disparity image. We have chosen to approach the problem by adding another term into the cost function in order to favour solutions with only small changes across scan lines. The cost function is now defined as

$$\begin{aligned} \gamma(M) = & N_{occ}\kappa_{occ} - N_m\kappa_r + \\ & \sum_{i=1}^{N_m} \Big(\underbrace{d(x_i, y_i)}_{\text{matching cost}} + \underbrace{\kappa_p |(x_i - y_i) - (x'_i - y'_i)|}_{\text{dissimilarity to previous line}} \Big), \end{aligned}$$

where N_{occ}, N_m is the number of occluded and matched pixels, respectively. The function $d(x, y)$ defines a sampling-invariant distance measure between position x on the current scan line in the left image, and position y in the right image (see Fig. 3). The matching sequence in the current scan line is denoted as $M = (x_i, y_i)$ and the minimum-cost matching sequence of the previous scan line is (x'_i, y'_i). The constants $\kappa_{occ}, \kappa_r, \kappa_p$ are determined empirically.

This approach is easier to implement and to tune than an additional postprocessing step. An example comparison between dynamic programming on independent scan lines to our modified, predictive cost function is depicted in Fig. 2.

4. VIEW INTERPOLATION

The view interpolation can be split into two parts: the translatorial motion of the camera, and the rotation. The translatorial motion induces parallax motion that can be compensated based on the estimated disparities. The rotational component is independent of the scene depth, and is simulated by gradually interpolating the rectification transforms.

Carrying out the compensation of parallax motion in the rectified images has the advantage that only simple horizontal pixel shifts are involved and sampling problems are

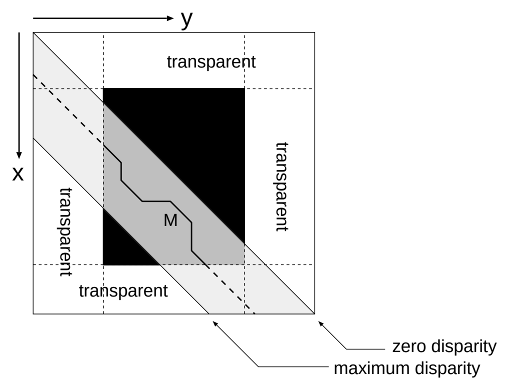


Fig. 3. Because the rectified image is non-square, each scan line has transparent regions at the left and right sides of the actual image content (black region). Disparity estimation is therefore restricted to the dark-grey area.

avoided. After compensating the translatorial part, the interpolated view is transformed back to cancel the rectification transformation (*un-rectification* step).

4.1. Interpolation of Rectified Views

For the interpolation between two rectified views, we assume that both views are available. However, this does not mean that every pixel in one view has a corresponding pixel in the other view. We can distinguish three cases.

A. *Pixels that are visible in both views.* These pixels have a corresponding disparity value d assigned to it. As can be easily derived, the pixel position in the intermediate view can be obtained by simply scaling the disparity by the inter-view position $v \in [0;1]$, giving the new disparity value $d' = v \cdot d$. We also interpolate the color value between the left and right view according to v. This case is indicated in Fig. 5 by double-headed black arrows.

B. *Pixels in occlusion-areas that are only seen in one view* (see marked areas in Fig. 4). These pixels have no disparity value assigned to it, because they were explicitly detected as occluded. However, as is easily visible from Fig. 4, occlusion pixels in the left view are occluded by objects positioned at their right side. Likewise, we know that occlusion pixels in the right view are occluded by objects at their left side. Assuming that occluded objects are parallel to the image plane, we extend the disparities of the background into the occlusion area. Hence, for pixels in the left view, the missing disparity values are copied from the left side, while for the right view, they are copied from the right side (see colored arrows in Fig. 5). Since these pixels are only visible in one view, we cannot interpolate their color values between the two views, but simply copy them.

C. *Pixels whose corresponding pixels are outside the valid image area in the other view.* For these pixels, no disparity was computed because of the restricted search-area

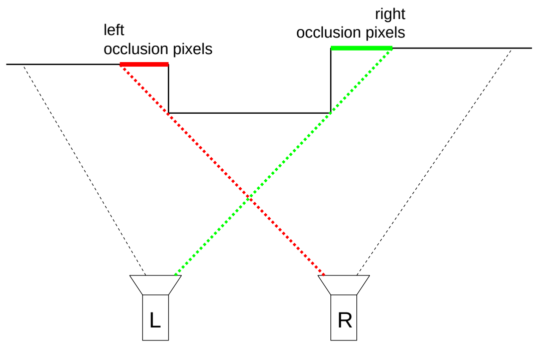


Fig. 4. A simple scene where a foreground object hides part of the background, inducing an occlusion region in each view.

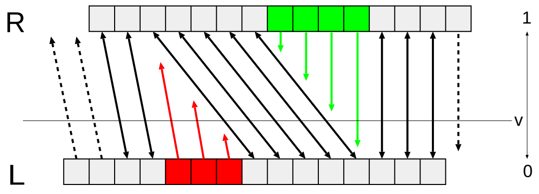


Fig. 5. Visualization of the interpolation process. Arrows indicate how pixels move in the parallax motion. An intermediate view corresponds to a horizontal cut at a position v between the two extreme views. Double-headed arrows indicate that the pixel color is interpolated between two views, while the data along single-headed arrows is simply copied.

(see Fig. 3). Again assuming that objects are mostly parallel to the image plane, we extend the disparity of the border pixels into the undefined region (see dashed lines in Figs. 3 and 5). Similar to the previous case, the pixel color cannot be interpolated, since the pixel is only visible in one view.

For a more complex view interpolation technique, we refer to [4]. This technique also takes into account the discontinuity between pixels interpolated from the two views and simple copied pixels, which could otherwise lead to color seams.

4.2. Un-Rectification

From the image-rectification process, we know the transformations between the original images and the rectified images for the extreme views. However, the transformation for the physically correct interpolation between these transformations would require strong calibration information, which is not available. Hence, we have to interpolate between both transformations such that we obtain a visually sensible motion. Simple interpolation of the transformation matrices $H_{(i)b}$, $H_{(i+1)a}$ usually leads to unnatural or even invalid transformations (mapping part of the image to infinity). As an alternative approach giving a visually pleasing motion, we consider the motion of the four corners of the rectified images. The position of these four corners are linearly interpolated and the intermediate transformation $H_{i(i+1)}^{(v)}$ is determined as the transformation that maps these four corners to the screen corners.

The un-rectification transformation is the last step prior to display, and it is hardware-accelerated using OpenGL. This results in real-time performance of the view interpolation at full SDTV resolution when executed on a 2.8-GHz Pentium processor.

5. RESULTS

An example interpolation result is depicted in Fig. 6. The input images were captured by a camera surrounding the objects on a circular path. Note that because the optical axes were not parallel, we have parallax motion in both directions. Even though our disparity estimation assumes only positive disparity, the situation can still be processed, because the rectified images are constructed such that all disparities are positive.

6. CONCLUSIONS

We have presented preliminary results of our ongoing work about free-viewpoint video. The described algorithm allows to interpolate intermediate views in a chain of cameras. Interpolation is possible between a set of still pictures or video sequences when the setup of the cameras is fixed. No explicit camera calibration is required and the decoding is carried out in real-time. Our future work will concentrate on improving the disparity estimation by employing an algorithm based on belief propagation [6], and using a back-projection view synthesis to allow for sub-pixel accurate shifts [5].

7. REFERENCES

[1] S. Birchfield and C. Tomasi. Depth discontinuities by pixel-to-pixel stereo. *Int. J. of Comp. Vision*, 35(3):269–293, 1999.

[2] R. I. Hartley. In defense of the eight-point algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(6):580–593, 1997.

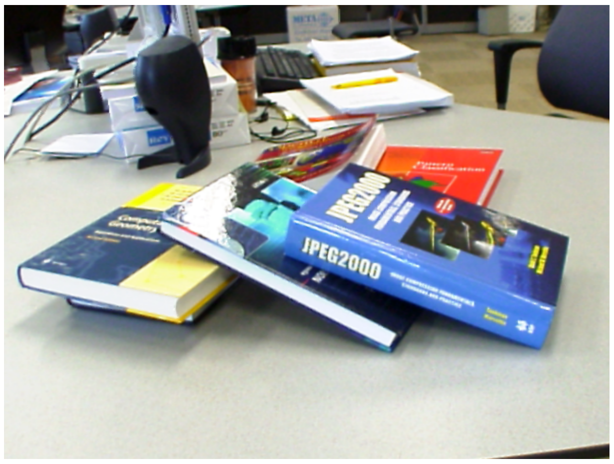
[3] R. I. Hartley. Theory and practice of projective rectification. *Int. J. of Comp. Vision*, 35(2):115–127, 1999.

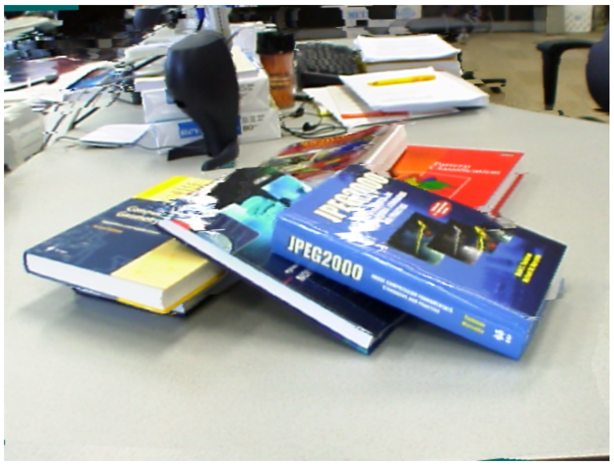
[4] E. Izquierdo and J.-R. Ohm. Image-based rendering and 3d modeling: A complete framework. *Signal Processing: Image Communication*, 15:817–858, 2000.

[5] M. Oliveira, G. Bishop, and D. McAllister. Relief texture mapping. In *Proc. SIGGRAPH 2000*, pages 359–368, July 2000.

[6] J. Sun, N.-N. Zheng, and H.-Y. Shum. Stereo matching using belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(7):787–800, July 2003.

[7] J. Xiao and M. Shah. Tri-view morphing. *Computer Vision and Image Understanding*, 96(3):345–366, 2004.


(a) left view


(b) interpolated middle view

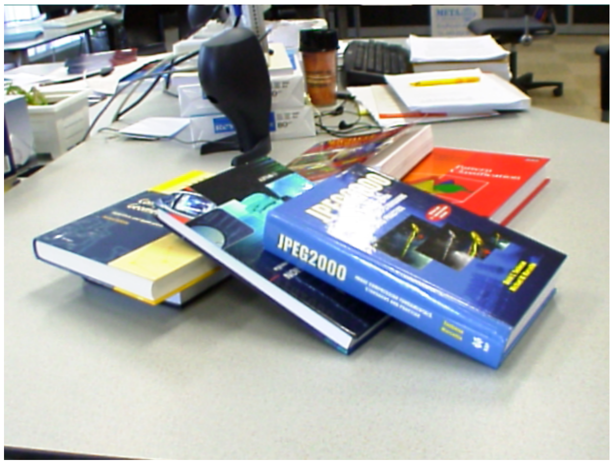

(c) right view

Fig. 6. Example interpolation result with large baseline. Note the large parallax of the chair on the right side and the specular lights on the books.