Hierarchical coarse to fine depth estimation for realistic view interpolation.

Indra Geys\textsuperscript{1} Luc Van Gool\textsuperscript{1,2}

\textsuperscript{1} ESAT / PSI-VISICS, Katholieke Universiteit Leuven Kasteelpark Arenberg 10, 3001 Leuven, Belgium igeys@esat.kuleuven.ac.be

\textsuperscript{2} D-ITET / BIWI, Swiss Federal Institute of Technology (ETH) Gloriastrasse 35, 8092 Zürich, Switzerland vangool@vision.ee.ethz.ch

Abstract

This paper presents a novel approach for view synthesis and image interpolation. The algorithm is build up in a hierarchical way, and this on different structural levels instead of using a classic image pyramid. First coarse matching is done on a 'shape basis' only. A background-foreground segmentation yields a fairly accurate contour for every incoming video stream. Inter-relating these contours is a 1D problem and as such very fast. This step is then used to compute small position dependent bounding-boxes in 3D space which enclose the underlying object. The next step is a more expensive window based matching, within the volume of these bounding-boxes. This is limited to a number of regions around 'promising' feature points. Global regularisation is obtained by a graph cut. Speed results here from limiting the number of feature points. In a third step the interpolation is 'pre-rendered' and simultaneously evaluated on a per pixel basis. This is done by computing a Birchfield dissimilarity measure on the GPU. Per pixel parallelised operations keep computational cost low. Finally the bad interpolated parts are 'patched'. This per pixel correction yields the final interpolated view at the finest level. Here we will also deal explicitly with opacity at the borders of the foreground object.

Keywords: 3-D from multiple images and video; Geometric signal processing; View Interpolation

1. Introduction

This paper presents a new hierarchical method to create accurate interpolated views of a dynamic scene. The input consists of the video streams and the calibration data from two or more static cameras. The output is a new video stream of a (moving) camera in between the inputs. Limited extrapolation is feasible as well. The typical environment will contain a moving person in front of an almost static background. The domain we focus on comprises tele-teaching and -conferencing applications with a higher level of automation. Illustrative examples are: the spontaneous selection of a frontal (interpolated) view of a speaker, a continuous transition from one camera viewpoint to another, etc. Computer aided -possibly automated- application of cinematographic rules [8] in combination with fast view interpolation are believed to open new possibilities for on-the-fly video editing and view generation.

View synthesis is a rapidly emerging field of research. Coming classically from user driven morphing algorithms [4] a gradual evolution towards more photo-realistic rendering of intermediate viewpoints is witnessed. Geometrically valid morphs between views of one static object were first proposed by Seitz \textit{et al.} [18] and Werner \textit{et al.} [19]. A second group of methods consists of light fields and lumigraph renderings [14, 11]. A large collection of 2D images are used to reconstruct a function that characterises the flow of light through the 3D space. Once this function is known, view synthesis can be performed in a straightforward way [17]. However obtaining, transmitting and processing such a very dense sampling of the environment renders these methods impractical for dynamic scenes. Generation and rendering of 3D models is also widely used. By using view dependent texture-mapping, as presented in the work of Debevec \textit{et al.} [7], textures are projected onto the geometry using a 'view map' for every polygon which improves visual quality.

A recent and even more challenging tendency is to generate interpolated camera's on-the-fly. The algorithms should now be designed to work on streaming video and computational cost becomes important. Quite some research has been done yet in the field of stereo matching without using dedicated hardware. Ansar \textit{et al.} [1] accomplish this by the use of bilateral filtering, while Yang \textit{et al.} [20] use the graphical board for computations. Assembly level optimisations using special extensions of the CPU instruction set (such as MMX) are also often used [12].

However very few algorithms for high speed view interpolation have been proposed so far. In [6] a fast interpolation of a stereo pair of an upper body is generated by dynamic programming, if the baseline is small. Zitnick \textit{et al.} [21] rather
focus on high-quality interpolations of pre-recorded videos of moving objects. Calculations are performed off-line. The visualisation of these nice intermediate views is done at interactive rates. In earlier work [10] we demonstrated how to use GPU-accelerated stereo for view interpolation. Recently Criminisi et al. [5] demonstrated that taking opacity effects into account can make the result much more realistic. Although the very good perceptual quality, this idea is computationally rather demanding. We will use an accelerated version of this approach, which does part of the computations needed for estimating the opacity, on the graphical board. The results will be coarser, but will still clearly improve the visible quality of the outcome video. Certainly at points where a virtual object is occluded by the foreground interpolation.

The next section 2 gives a general outline of the algorithm. Section 3 presents a concise overview of the preprocessing. Sections 4, 5 and 6 respectively describe coarse, intermediate and fine level matching and result generation. Experimental results are shown in section 7, and section 8 concludes the paper.

2. New view synthesis

Our new view synthesis algorithm is composed of 3 levels of matching (see also figure 1). Important is that we focus on generating realistic interpolations, and not on an accurate 3D reconstruction of the complete environment. Cameras are assumed to be calibrated on beforehand.

First, we ‘pre-process’ the video-streams. This comprises correction for lens distortions by the GPU and removal of the global colour mismatch between the different videos. Afterwards foreground-background segmentation is performed. ‘Coarse matching’ relates the shapes of the contours of the foreground object, computed from the segmentation result. After filling in missing and erroneous matches, position dependent bounding-boxes which enclose the 3D object are computed.

The second step is a more accurate, ‘region based correspondence search’. Windows around a set of promising 2D foreground points are correlated. A graph-cut algorithm solves the depth estimation in a global way. Finally, artefacts in the result are detected and corrected using a Birchfield dissimilarity (accelerated by using the GPU) on ‘pixel level’. The correction is twofold. First we check if a refined depth estimate lowers the error after re-projection. If not we assume a colour mismatch due to transparency effects and adapt the local opacity of the object. This typically occurs at the border of the foreground. The current implementation of the algorithm can reach approx 5 fps for 640x480 images, at a Pentium IV@2.6 GHz.

3. Video preprocessing

Radial distortions (see fig. 2) are common for the low-end lenses of the cheap cameras we use. The distortion parameters are lens specific and are determined during camera calibration. The actual unwarping of the incoming frames is performed by the GPU by projection onto an inverse distorted planar surface. The use of display lists to batch commands, minimises the overhead by driver calls.

Colour correction at this stage will be performed by the parameters estimated for the previous frame. How this estimation is done is explained in section 5.3. The higher the frame rate at which we can process, the better this historic parameters will normally be.

Segmentation is done for all incoming video streams. A fast algorithm that is resistant to local illumination changes is developed based on [16]. Foreground and background are regarded as two different layers, for which the depth and the interpolation are obtained separately. In this paper, we focus on the foreground object. Interpolation of the background can be done in a similar way or can be updated at a lower rate, using a more expensive (off-line) algorithm. This is
justified by the fact that the backgrounds are quasi static. The segmentation result for the background is also used to continuously update the background reference image. By this we will gain robustness for a slowly varying environment and we get an updated texture at frame rate, despite the underlying geometry which can be updated slower.

4. Coarse matching

The segmentation algorithm provides us with a binary image (see also fig. 3). A chain of image pixels representing this foreground contour can be extracted very fast by using MMX-optimised routines. As the cameras are calibrated, the epipolar geometry between neighbouring camera pairs is known. With \( F_{LR} \) we will denote the fundamental matrix between cameras \( L \) and \( R \). The \( n \)-th pixel on the contour in camera \( L \) is indicated by \( C_L(n) \). The corresponding epipolar line for camera \( R \) is \( l_{epiR}(n) = F_{LR} C_L(n) \).

No texture information can be used for matching the contours, since ‘corresponding’ contour pixels don’t necessarily originate from the same point in 3D. This can be both due to a slightly different segmentation, or to the possibly large differences in camera viewpoint. A four step procedure is used to ‘match’ the contours between two consecutive camera pairs.

- **compute candidate matches:** for every point of the contour, intersect the corresponding epipolar line with the contour in the next camera: \( \forall n \in C_L(n) \) compute \( l_{epiR}(n) \cap C_R \)

- **select good matches:** given the limited depth range of most foreground objects, it is valid to assume that monotonicity is preserved. This means that the first and the last intersection of the epipolar line with the contour in both images can be considered a match. In the example of fig. 3 this results in the tuples \( \{a, a'\} \) and \( \{d, d'\} \). All resulting matches are indicated in the middle figure, with both contours superimposed. This step will match all convex parts of the contour.

- **interpolate good regions:** for non convex parts of the contour (e.g. the border between the arm and the body) a match is predicted by interpolating the matches of the neighbouring convex regions. These will always exist since the contours are closed. The prediction is replaced by the closest candidate computed in step 1.

- **remove erroneous matches:** the disparities of corresponding points \( p = C_L(n) \) and \( p' = C_R(n) \) must be similar as to their directions and magnitudes:

\[
\angle p, p' \approx \sum_{\# matches} \text{direction} \quad \& \quad \| p - p' \| \approx \sum_{\# matches} \text{length}
\]

Although working on one-dimensional data structures, the first two steps require rather much contour transitions, and the complexity grows quadratically with the length of the contours \( O(n^2) \). This can be overcome by generating a lookup-table (lut) to transform an image in its rectified counterpart. The image is not rectified, but during a first transition every contour point will be added to a small list of points for the corresponding epipolar line (corresponding to the height of the rectified point). The lut makes this operation very fast. During a second contour transition the generation of candidate pairs and good matches becomes trivial by checking the corresponding (sorted) list for each contour point. This reduces the complexity back to \( O(n) \) at the cost of extended memory use to store the luts, which are computed at calibration time.

Now we will use the matched contours to generate a position dependent bounding-volume in 3D space. By back-projecting the corresponding contour points along their camera rays we get intersection points in 3D. This is a bit similar to shape-from-silhouette techniques \([13, 15]\). It is however not our objective to compute a complete visual hull, as this would require more cameras and involves rather expensive polygon intersection algorithms. We are merely interested in determining a fairly accurate depth range covering the different parts of the foreground object, which will be refined later on.

As stated before, the ‘matches’ computed will mostly not originate from the same 3D point of the object. Camera rays in different views will be tangent to the object at different points. However, the rays through the contourpoints...
always intersect each other outside the volume spanned by the object. Fig. 4 shows one 'slice' of the scene of fig. 3 from a viewpoint perpendicular to the image plane.

![Diagram of a graph with labeled nodes and edges](image)

**Figure 4.** 'Corresponding' contour points in two images not always originate from the same 3D-point. The intersections of their rays determine a point on the bounding-box around the object.

Not only the rays of corresponding contour points should be intersected in 3D space, as this would not give a bounding-volume but simply the shape of the contour in space. We will reuse the sorted lists for every scanline to speed up the matching process. If we take a closer look to an individual scanline, it is noted that in case of an even number of intersections with the contour, every two subsequent contour-points ({a, b} and {c, d}) cover a part of the foreground, intermediate tuples ({b, c}) cover background. If we intersect all rays of the foreground-tuples we get a local bounding-box in space for this part of the foreground object. This is illustrated in fig. 4 by the dark points, which form the bounding polygon. Note that this approach takes explicitly into account our assumptions about the foreground, and results in a much more reduced bounding-volume than what would be obtained by the visual hull (in fig. 4 this would be the volume obtained by intersecting all projection rays). The computations are also easier and as such faster. In case an odd number of intersections, the epipolar line is tangent to the contour. A similar strategy can be used.

### 5. Region based matching

The knowledge of only a bounding-volume of the foreground object is not sufficient to generate new views. A more accurate depth search is performed within this bounding-volume. To overcome 'line shaped' artefacts -typical for scanline based approaches- we minimise an energy, containing a spatial coherence term which relates pixels both in the direction and across the scanlines of the image. A min-cut algorithm [3] is used to solve this energy minimisation problem and regularises the depth search. It finds the 'cutting surface' through a graph which minimises the cost of the broken interconnectivities. To limit the computation time, only a set of regularly distributed points, together with the contourpoints are used to set up the graph.

#### 5.1. Min-cut of a graph

The weighted graph consists of layers of nodes which represent the different depth levels. Inside such a layer, the nodes are interconnected by four edges representing the spatial coherence term. The weights of these edges are based on the similarity of the neighbouring intensity values. Every node is also connected to its two neighbouring layers. These edges are weighted according to the correlation measure between the downsampling input images. For this the Sum of Absolute Differences (SAD) is used, which for multiple cameras is defined as follows:

\[
SAD(x, y) = \sum_{l = 0}^{NC} |I_l(x, y) - \sum_{l = 0}^{NC} I_l(x, y)| \quad \text{with} \quad l \in \{0, 1, ..., NC\} \quad \text{and} \quad NC \text{ the number of cameras.}
\]

A cut through the graph partitions the nodes into two disjoint subsets. As such the cutting surface appoints a depth value to every point. Solving the min-cut problem finds the depth map according to the energy to be minimised (see fig. 5 [right]). For more details see [10].

#### 5.2. Point sampling and mesh generation

Points sampled on a regular grid give the best distribution over the foreground. However, a uniform sampling doesn’t take into account any image properties. Points in non-textured regions will not be well conditioned for matching and a local correlation-based approach will not be able to define a clear depth estimate. Therefore we adapt the sampling by the location of the edges in the image. Points on edges are expected to be more robust for matching. Edges can be caused not only by the texture but also by depth differences. In the last case also the different depth areas will be delineated better by taking the samples at the edges.

For fast edge detection, a Sobel filter is used. A neighbourhood is examined around every sample position on the regular grid. If an edge is found within this region, the sample is shifted from its position on the grid towards the edge. An example of the resulting sampling is given in fig. 5. A triangle mesh is created through the sampled foreground points and the contour points (see fig. 5 [middle]). This is performed by an MMX-optimised Delaunay triangulation.

#### 5.3. Colour scale factor calculation

Colour differences appear in images taken by different cameras from a different point of view. These differences arise from slightly different illumination conditions and the non-linearity of the camera hue capture. A correction is needed to perform consistent image blending and better correlation based matching.
A global linear colour transformation is used to do this correction. We don’t take cross-talk between the different colour bands into account, but treat every band separately. This delivers a satisfying correction and is more robust for misfits of the colour transformation. Three scale factors need to be determined, one for each colour band.

The depth map resulting from the graph-cut delivers us point correspondences between all inputs. The correspondences between two images are used to relate the colours of those images. The image taken by the camera closest to the desired viewpoint is taken as a reference image for this colour transformation. The colour values of the other inputs are scaled as to correspond to the colour of this reference. Not all correspondences are scaled as to correspond to the colour of this reference. The colour values of the other inputs are scaled as to correspond to the colour of this reference. The remaining \( N^* \) ‘good’ colour correspondences \( \{ P^{K*}_i, P^{R*}_i \} \) are used to calculate the colour transformation. Outliers are catered for by applying a RANSAC [9] approach. After the removal of these last \( Nb \) spurious colour matches a linear least squares problem of the form \( A_c x_c = B_c \) is solved for every colour band, with \( A_c = [ P^{K*}_c, P^{K*}_c, \ldots, P^{K*}_{c(N^*−Nb)} ]^T \) and \( B_c = [ P^{R*}_c, P^{R*}_c, \ldots, P^{R*}_{c(N^*−Nb)} ]^T \). The scale factor \( x_c \) has to be determined. This new transformation is applied to all images and results in a better colour match between them, increasing the performance for the remainder of the pipeline. The same colour transformations will also be applied to the next frame at the beginning of the pipeline, as explained in section 3. Fig. 6 illustrates such colour scaling.

6. Pixel level refinement and new view rendering

Using triangulation, the depth values calculated by the graph-cut are linearly interpolated per triangle, yielding a 3D coordinate for every pixel. However, this linear interpolation introduces artefacts, for example in the proximity of sudden depth discontinuities. Not taking opacity into account can also lead to visually unappealing results, especially at the borders of the object. During the creation of an interpolation, artefacts are detected and corrected by measuring the Birchfield dissimilarity.

6.1. Rendering and artefact detection

Using the depth values obtained by the graph-cut algorithm, the triangle mesh (section 5.2) can be rendered in 3D. A linear interpolation of the depth values inside the triangles occurs. To obtain a realistic interpolation, the input textures from the two cameras closest to the desired position are projected on the object. A blending factor dependent on the relative distance from the virtual camera to these two real cameras is applied. Fig. 7 illustrates the result. If the depth estimate is wrong, the textures don’t match perfectly and the blending generates a blurry result. To detect these bad patches, the GPU calculates the Birchfield distance [2] between both projected images during rendering.

The Birchfield distance \( D(x_L, x_R) \) is a pixel dissimilarity measure that is insensitive to image sampling. Consider two corresponding pixels \( x_L \) and \( x_R \), from a left and a right image. First the linear interpolated intensities \( I_L^- \) and \( I_L^+ \) halfway between \( x_L \) and its neighbours are determined, as illustrated in figure 8. Let \( I_{Lmin} = \min(I_L^-, I_L^+, I_L^0) \).
and $I_{L_{\text{max}}} = \max(I_L^{-}, I_L^{+}, I_L^{0})$ (see fig. 8), then $D(x_L, x_R)$ is defined as follows:

$$D_1 = \max(0, I_L^{0} - I_{R_{\text{max}}}, I_{R_{\text{min}}} - I_L^{0})$$

$$D_2 = \max(0, I_R^{0} - I_{L_{\text{max}}}, I_{L_{\text{min}}} - I_R^{0})$$

$$D(x_L, x_R) = \min(D_1, D_2)$$

Using the latest graphical hardware, multiple textures can be rendered simultaneously and programmer defined shading operations can be executed during rendering. We implemented the pixel-wise calculation of the Birchfield distance as a fragment program.

In a first rendering step, $I_{L_{\text{min}}}$ and $I_{L_{\text{max}}}$ are determined. Three images are orthographically projected on a plane. These three images are the intensity image $I_L$, $I_L$ shifted with $\frac{1}{2}$ pixel to the right and $I_L$ shifted with $\frac{1}{2}$ pixel to the left. As such, the intensity values $I_L^{0}$, $I_L^{-}$ and $I_L^{+}$ are available in the different images at the same pixel position. The GPU computes the pixel value interpolation. A fragment program is implemented to calculate the values $I_{L_{\text{min}}}$ and $I_{L_{\text{max}}}$ for every position. The result available in the frame buffer consists of an rgb-colour texture composed of $I_L^{0}$ in the red band, $I_{L_{\text{max}}}$ in the green band, and $I_{L_{\text{min}}}$ in the blue band. A similar composite texture is created for the right input $I_R$. In a second step both composite textures are used to calculate the Birchfield dissimilarity $D(x_L, x_R)$, according to equation 1. This can be realised by a fragment program as well. The calculations are performed while projecting the textures onto the 3D model. We output the dissimilarity values in the alpha-channel of the result, obtained in the frame buffer. The same fragment program can be used to (simultaneously) blend the original input textures on the object, since four textures can be rendered simultaneously. This colour interpolation is obtained in the rgb-channels of the frame buffer. The pixels where the Birchfield distance is above a threshold, are categorised artefacts and need further examination. They are shown in white in fig. 7[right].

6.2. Artefact Removal

The artefacts detected by the Birchfield distance can originate from two underlying problems. The depth estimation can be wrong or the foreground object can be locally (semi-)opaque. Both of these problems can be solved as to correct the interpolation result.

6.2.1. Refined depth search. For all the pixels where a high Birchfield distance is measured, we do a new local depth search. This is performed by a plane sweep algorithm, implemented on the GPU. A local planar patch, parallel to the image plane of the desired camera, is swept through 3D space in a small range $[D_{\text{min}} - D_{\text{max}}]$ around the initial depth value, obtained from the graph-cut. The input textures are projected on the patch and the pixel-wise Birchfield distance is calculated by a fragment program. These distances are compared at every depth and only the best ones, together with their corresponding depth and colour values are retained in the final result. The colour value obtained for the pixel under consideration is used to correct the interpolation. For all pixels where a small depth error occurred, this refinement technique yields satisfying depth values and more accurate colour values in the interpolated view. Restriction of the refined depth search to a small range $[D_{\text{min}} - D_{\text{max}}]$ is valid in the assumption that no large depth differences occur locally in the foreground. As such the depth values can be determined more accurately on these places where is it necessary.

6.2.2. Opacity calculation and patch composition. At (semi-)transparent points of the foreground the refined depth search still can’t provide a solution for the artefacts in the interpolation. The background is (partially) visible through the object. Since the input images are taken from

![Figure 7](image1.png) [left] Triangle mesh rendered and lit by OpenGL, seen from an intermediate position. [middle] Textured triangle mesh. Artefacts are visible around the contour and at the eyes. [right] The Birchfield dissimilarity is considered too high in the white regions. The interpolation should be improved.

![Figure 8](image2.png) Figure 8. Calculation of the Birchfield distance between two intensity images $I_L$ and $I_R$. 
different locations, the background colours behind the object differ, as will the colours captured by the cameras. This results in a high Birchfield dissimilarity.

The composition of the colour value $C_i(x, y)$ captured by camera number $i$ can be written as follows:

$$C_i(x, y) = \gamma \ast F(x, y) + (1 - \gamma) \ast B_i(x, y)$$  \hspace{1cm} (2)

where $\gamma$ is the opacity value, $F(x, y)$ the pure foreground colour and $B_i(x, y)$ the local background colour. Dependent on the viewing direction of the camera, other background colours $B_i(x, y)$ can contribute to the colour captured.

For the creation of an intermediate view of a (semi-) transparent object, it is necessary to determine $\gamma$, $F(x, y)$ and $B_i(x, y)$ and use equation 2 to composite the correct colour.

It is not correct to interpolate the colour values of corresponding points in the input images since they contain contributions of different background parts.

A reference background image is used for our foreground segmentation (see section 3). Since it is updated every frame to correspond to the new incoming colours, it can deliver the $B_i(x, y)$ values. By generating a background interpolation also the local background colour $B_i(x, y)$ for each foreground point in the new interpolated image is known.

The pure colour $F(x, y)$ and the opacity values $\gamma$ are properties of the foreground object itself and as such independent of the acquisition position. For a point of the interpolated image, the corresponding points in the input images can be determined. Since we know the captured colour $C_i(x, y)$ and the background colour $B_i(x, y)$ for the corresponding points in two or more input cameras, we obtain a set of equations (eq. 2) in two unknowns. As such $\gamma$ and $F(x, y)$ can be determined and the correct intermediate colour can be composed according to equation 2. These calculations are done pixel-wise.

7. Results

We obtain fairly accurate interpolations of a scene, as shown in fig. 9. These interpolations are obtained by using three input cameras. They are results at one point in time, with the virtual camera moving in between the inputs. Our algorithm works well, given even rather wide baselines. Moreover, the wider the baseline the better the bounding-volume encloses the underlying geometry and the smaller the search range for the region based matching. This results in an interesting trade-off since a smaller baseline yields a larger bounding-volume, but also a better conditioned region based matching.

Fig. 10 illustrates the importance of the depth refinement. When taking a closer look at the face, the difference in accuracy becomes clear. Details become much sharper. By taking the opacity values into account, the transition to the background becomes better. The disturbing border, resulting from an inaccurate segmentation or transparent foreground parts, is suppressed.

The importance of including opacity information becomes especially clear when the foreground object is placed in front of a virtual background. The person in the right image of fig. 11 seems to fit more naturally in the new environment.

Finally, we want to visualise the effect of colour correction. In fig. 12 an interpolation is rendered, before any depth refinement is performed. Two input textures are projected on the model, but they are not blended. Instead a chessboard...
pattern is used to visualise both of them. The colour differences after applying the colour transformation are less salient.

8. Conclusion

A new hierarchical method is developed to create realistic interpolated views of a scene. The input consists of video streams and the calibration data of multiple cameras. First a coarse contour matching is done to determine a bounding-volume around the foreground object. As a second step a region based matching is performed inside this estimated depth range. Global regularisation is obtained by determining the min-cut through a graph, which is created by sampling ‘good’ points. Colour adjustment from all inputs to one reference image increases the similarity between the images. As a last step, remaining artefacts are detected by calculating the Birchfield dissimilarity on the GPU. These artefacts are removed by a refined depth search and the estimation of the foreground opacity.

Future work will focus on further speeding up the process, such that also non-static backgrounds can be dealt with.

Acknowledgements

The authors gratefully acknowledge support from the Research Fund K.U.Leuven (GOA).

References


Figure 9. A sequence of interpolations obtained by a ‘virtual’ camera moving between three input cameras.