REAL-TIME DEPTH MAP GENERATION ARCHITECTURE FOR 3D VIDEOCONFERENCING

John Congote[1,4]*, Iñigo Barandiaran*[1]*, Javier Barandiaran*[1]*, Tomas Montserrat*[2]*, Julien Quelen*[2]
Christian Ferran[2]*, Pere J. Mindan*[3]*, Olga Mur*[3]*, Francesc Tarres*[3]*, Oscar Ruiz*[4]

[1]Vicomtech
Research Center
San Sebastian – Spain

[2]Telefonica Research
Barcelona – Spain

[3]UPC
GTAV Group
Barcelona – Spain

[4]EAFIT University
CAD CAM CAE Laboratory
Medellín – Colombia

ABSTRACT

In this paper we present a reliable depth estimation system which works in real-time with commodity hardware. The system is specially intended for 3D visualization using autostereoscopic displays. The core of this work is an implementation of a modified version of the adaptive support-weight algorithm that includes highly optimized algorithms for GPU, allowing accurate and stable depth map generation. Our approach overcomes typical problems of live depth estimation systems such as depth noise and flickering. Proposed approach is integrated within the versatile GStreamer multimedia software platform. Accurate depth map estimation together with real-time performance make proposed approach suitable for 3D videoconferencing.

***Index Terms* —** depth map, stereo vision, 3D videoconferencing, autostereoscopic displays, stereo matching

1. INTRODUCTION

During the last years, videoconferencing has become a widely extended application. Companies and individuals use this technology to reduce transportation costs and to improve communication between distant parties. However, traditional 2D videoconferencing still fails to produce natural impressions to the remote conferees. 3D videoconferencing is a logical evolution: a better feeling of presence is provided to conferees by leading them to believe that they are closer to each others. Currently, the emerging 3D displays and the increase of mainstream hardware computation capabilities make telepresence feasible.

Our contribution is a 3D videoconferencing approach which implements a visually accurate stereo algorithm running over GPU. The outline of the paper is as follows: In section 2, a short review of different approaches and techniques related with 3D videoconferencing is presented. Methods and algorithms implemented in our approach are shown in 3. Then, an architecture where proposed approach is validated is described in section 4. Finally, we detail some conclusions and suggestions for future work in section 5.

2. RELATED WORK

Many important topics related to videoconferencing must be addressed to enhance user experience. One of such issue is eye-contact or gaze. Research [1, 2] has shown that gaze is one of the most important non-verbal cues, responsible for giving feedback or expressing feelings and attitudes. Another problem related with videoconference is the lack of perception of depth wrt the analyzed scenario. In a 2D videoconference, traditional displays are used. The absence of depth information in this type of displays, in addition to the problem of gaze results in an unnatural experience.

An accurate and efficient depth map estimation mechanism is needed for both problems to be addressed. For example, depth information is used by autostereoscopic displays to render multiple novel views. These generated views can be used to tackle the problem of eye-contact, by generating views that agreed with the user eye-sight, as in [3]. Moreover, autoestereoscopic displays typically use depth maps information to generate 3D perception, [4](See Figure 1).

In [5] Scharstein presents a set of stereo matching methodologies used to calculate dense disparity maps from stereo images, which can be classified in local, global and semi-global algorithms. The depth calculation process is done by identifying correspondences between pixels in two images. Given the location of corresponding pixels in the images, their 3D coordinates can be retrieved by means of triangulation. Applications like 3D Videoconferencing imposes strong time restrictions, thus only real-time methods are suitable.

Figure 1. 2D+Depth image result of our system

3. METHODOLOGY

In this section we describe in detail the methods implemented in our architecture, which allow us to achieve accurate depth information in real-time.

3.1. Image Rectification

Depth map calculations rely on the estimation of relative pixel positions between two images. The position of the cameras and lens distortions make necessary some previous corrections. These corrections are carried out before depth map calculation takes place in

978-1-4244-6379-4/10/$26.00 ©2010 IEEE

order to improve the process efficiency. Moreover, it is also necessary to align the horizontal lines of both images in order to reduce the search process of the pixels to a 1D problem. The tools needed to carry out this transformation are the camera calibration parameters, which should have been calculated previously. Results of this process can be observed in Figure 2.

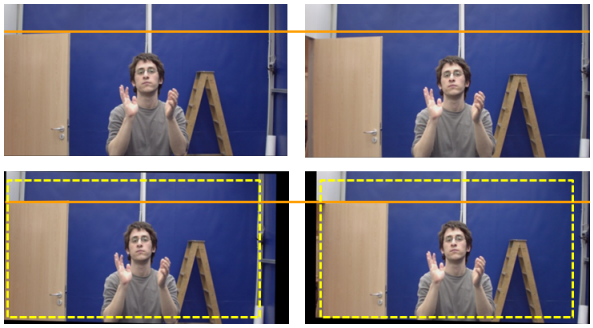

Figure 2. Top: Left and right captured images. Bottom: Undistorted and rectified images, yellow frame denotes cropping area.

Although camera calibration can be carried out just once and off-line, rectification should be done online and for each pair of acquired images. As we work with large images, we need a fast and effective algorithm. The solution adopted in our approach is the use of GPU algorithms implemented in CUDA. Our implementation processes the pixels in parallel, taking advantage of the multicore architecture of the GPU. Execution of our algorithms on the GPU outperforms General Purpose Processors (GPP) based solutions and obtains real-time performance at a reasonable cost. Rectified images have a resolution of 960×540 pixels as required by our 3D display. After undistortion and rectification processes, some points of the image may lay outside the working frame, producing some empty parts (see Figure 2). In order to eliminate these unwanted regions, input images are captured at higher resolution (1200×600) and cropped after rectification process takes place as seen in Figure 2.

3.2. Adaptive support-weight

Adaptive support-weight *AW* is a local algorithm to calculate dense stereo disparity maps, presented by Yoon[6]. The algorithm is a window based method for correspondence search using varying support weights. The support weights of the pixels in a given support window are based on a combination of color similarity and geometric proximity to reduce ambiguity. *AW* gives a cost to a pair of pixels depending on their support windows, representing how related the pixels are. Afterwards, each pixel is correlated to the pixel that has the associated minimal cost in the matching set.

A dense disparity map can be represented as an image D of the same size of the input rectified images L, R. Where $D_{i,j}$ represents the displacement of the coordinates of a given pixel from L to R, consequently $L_{[i+D_{[i,j]},j]} \approx R_{[i,j]}$. Image pixels p, q are commonly represented as an *RGB* 3-tuple then p_r is the *Red* component of the pixel p as p_g and p_b the *Green* and *Blue* components, hence the color distance between two pixel is defined as (eq: 1, 2) following the L_1 and L_2 distance norms.

$$f_c(p,q) = |p_r - q_r| + |p_g - q_g| + |p_b - q_b| \tag{1}$$

$$g_c(p,q) = \sqrt{(p_r - q_r)^2 + (p_g - q_g)^2 + (p_b - q_b)^2} \tag{2}$$

Equation 3 shows the m image (left or right) weight contribution of the pixel in the position (k,l) in a window centered at coordinates (i,j). The γ_c and γ_s values are empirical factors that modify the color and space similarity respectively. In our implementation, we set these factors to $\gamma_c = 20$ and $\gamma_s = 1$. The γ_c factor depends on how close the values in the *RGB* color space are, while γ_s modifies the space similarity, which is closely related to the support window size.

$$f_w(m,i,j,k,l) = e^{\left(-\left(\frac{g_c\left(m_{i,j},m_{k,l}\right)}{\gamma_c} + \frac{\sqrt{(i-k)2+(j-l)2}}{\gamma_s}\right)\right)} \tag{3}$$

Calculation of the disparity map D for each pixel $D_{i,j}$ in the reference image L is made by the *Winner Takes All* method as show in (eq: 6), where d represents the range of values where the correlation of pixels will be tested. This value depends on the camera parameters and the depth search range defined, furthermore the parameter w presented in (eq: 5) indicates the size of the square support window for pixel correlation. Finally SW (eq: 4) is a component used in (eq: 5) representing Support Weights used to influence the difference of the color and position in the window (eq:1). Notice that the cost support weight for a given pixel in L depends on the matching candidate, so normalization is required in (eq: 5).

$$SW := f_w(L,i,j,s,t) f_w(R,i+d,j,s+d,t) \tag{4}$$

$$f_{aw}(L,R,i,j,d) = \frac{\sum_{s=i-w}^{i+w} \sum_{t=j-w}^{j+w} SW\, f_c(L_{s,t}, R_{s+d,t})}{\sum_{s=i-w}^{i+w} \sum_{t=j-w}^{j+w} SW} \tag{5}$$

$$f_d(L,R,i,j) = \arg\min_{d \in \mathbb{Z}} f_{aw}(L,R,i,j,d) \tag{6}$$

Although *AW* is a local stereo matching algorithm, real-time performance is difficult to achieve with CPU implementations. However, the local nature of the algorithm allows an easy implementation in parallel architectures such as GPUs, achieving real time performance. We use a support window of 1×15 in size and only for the reference image, a maximum disparity of 30 pixels and an image size of 480×270 pixels. With this specifications we can calculate the depth map for both images in 8ms. Note that the depthmap is processed at a quarter of the resolution of rectified images. Post-processed depth map is upscaled afterwards to match the reference image resolution. Mentioned modifications to the original AW algorithm caused a loss of quality in the resulting depth map. However, our proposed post-processing technique is able to correct most of the wrong matches introduced by the algorithm simplification while still keeping real-time performance. Cross-checking is a common process for identifying unreliable pixels in the calculated depth maps. Matched pixels in left and right disparity maps are tested for consistency, where inconsistent matches are discarded. These pixels are later approximated by an anisotropic diffusion post-processing step. Even though regions with no texture cannot be correctly calculated, it is assumed that these regions are part of the background which is removed from our depth map. This background segmentation process is obtained using the statistical method described in [7], also implemented on GPU. The integration of a segmentation process allows us to obtain a more accurate depth map even on untextured regions, avoiding uncomfortable flickering effects during the conference.

3.3. Post-Processing

As mentioned before, cross-checking is used in our system to detect wrong correspondences by building a mask with them. This mask also includes information regarding foreground segmentation. The main objective of post-processing is to assign correct disparity values to all unreliable pixels detected in the previous step. Moreover, the disparity map is improved by removing the quantization effect introduced by the lack of sub-pixel accuracy during disparity estimation.

Mismatches basically occur in occluded regions as well as in homogeneous texture areas. A common hole-filling criterion is that occlusions must not be interpolated from the occluder, but only from the occludee to avoid incorrect smoothing of discontinuities. Thus, an extrapolation of the background into occluded regions is necessary. In contrast, holes generated due to mismatches can be smoothly interpolated from all neighboring pixels. If we assume that discontinuities in both disparity and color images are co-aligned, the previous interpolation criterion can be achieved by relying only on reference image color cues.

We use a variation of the classic Perona-Malik anisotropic filter [8] in order to propagate correct disparity values through the image. Anisotropic diffusion is a non-linear smoothing filter. It produces a Gaussian smoothed image, which is the solution to the heat equation, with a variable conductance term to limit smoothing at edges. For this particular implementation, the conductance term is a function of the reference image gradient magnitude at each pixel and for each color channel. As we can partially rely on foreground segmentation information, unreliable pixels are initialized to their known background disparity value. Another difference between standard Perona Malik discrete filter implementation and our approach is that in initial iterations we only modify values of unreliable pixels, avoiding them to influence on correctly assigned disparities. These initial iterations are intended to fill the disparity map holes. Subsequently, after convergence of discarded pixel values, we apply a reduced number of additional diffusion iterations modifying all pixels (see figure 3). This second diffusion step is intended to remove the quantization effect of the disparity map. This post-processing algorithm also allows an efficient and easy implementation in GPU architectures.

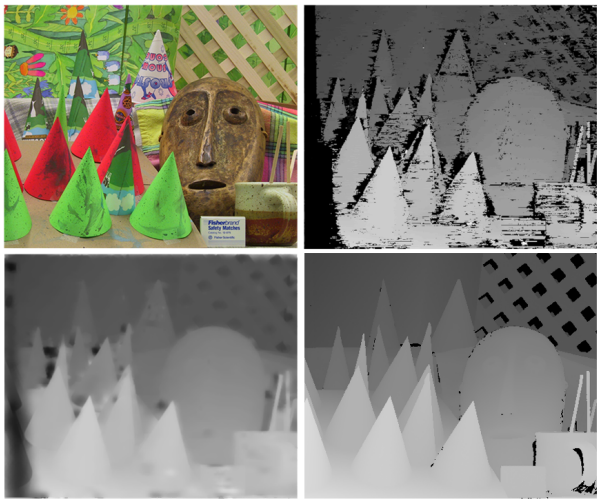

Figure 3. Post-Processing (from left to right and up to down): Reference Image. Initial Disparity Map (Black pixels are wrong pixels). Final Disparity Map. Ground truth

4. RESULTS

Current 3D displays are capable of providing multiple stereoscopic views in order to support head motion parallax. Hence, the user is able to perceive depth when uses these types of displays. This 3D perception can reinforce the feeling of volume and therefore the feeling of "tele-presence". In our prototype, we are using a 42" Wow display designed by Philips. This display can generate several views of the same scene, by using a 3D data interface in the form of image-plus-depth with a resolution of 960×540. This input format was standardized by MPEG and is currently implemented in many commercial 3D displays. This format proposes the use of images that are double the size of the native resolution, where the reference frame is disposed in one side of this image, and the corresponding disparity or depth map on the other, as depicted in figure 1. In order to be able to estimate a disparity map, two or more images of the same scene are needed. In our prototype we use two Pixelink digital cameras for stereo image acquisition. These cameras capture images at a resolution of 1200×600 pixels and with 25 fps framerate. We use RAW Bayer native image format, and compute Bayer demosaicing and white balance, inside the GStreamer pipeline. By using RAW images the required bandwidth is reduced, optimizing memory usage and allowing better off-camera Bayer demosaicing.

4.1. Integration Pipeline

Our streaming architecture is based on GStreamer software[9]. This library is an open source framework for creating streaming media applications. The design of the framework is based on modularity, where any process in the streaming pipeline can be encapsulated in an independent module or plug-in. This modularity provides flexibility and allows the user to easily test different configurations and approaches. Moreover, threading and buffering architectures are controlled. Our architecture is divided in two separated parts. One of these parts works as the content generator or server, where the videoconference content, i.e, 3D images for visualization, is generated. The second part of the architecture represents the client side, where these images are received and displayed to the conferees. In the following section, we describe in more detail both parts of the proposed architecture.

4.1.1. Server Side

This part of the architecture runs as a content generator. It manages the digital cameras that capture the scene, the depth estimation and encoding modules. Cameras are externally triggered by using specific hardware, in order to have a perfectly synchronized image acquisition. This synchronization significantly increases the accuracy of disparity estimation. Figure 4 shows the complete processing pipeline integrated into the server side of the architecture. The core of the pipeline is the plugin called *Depthestplugin*. This plugin receives two video streams from two synchronized threads, and executes image rectification and depth map estimation procedures. The output of the plugin is composed of a reference image and the corresponding disparity map. These streams are queued in two buffers before encoding them by using the h.263 video compression format. Finally, encoded video streams are rtp-packetized and sent through udp socket.

4.1.2. Client Side

This part of the architecture runs as a content receiver. It represents the end point of the entire pipeline, where the output device,

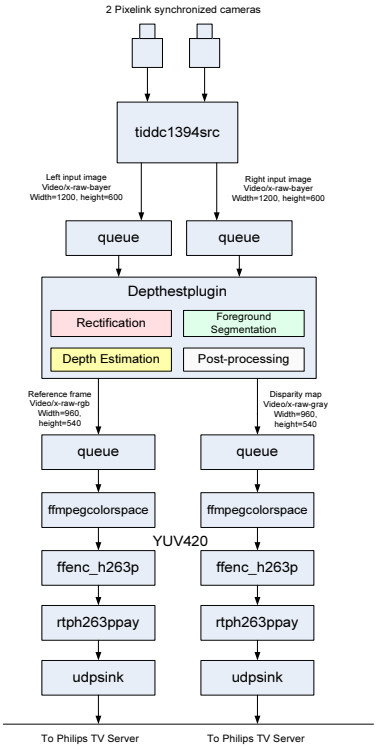

Figure 4. GStreamer 3D videoconference Server Side

i.e. the autostereoscopic display, is active. As shown in Figure 5, the first part of the processing pipeline consists of the network and video decoding of both streams. Before the video content can be

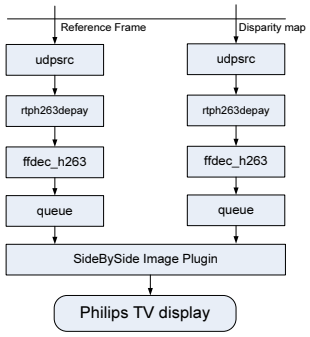

Figure 5. GStreamer 3D videoconference Client Side

visualized on the display, it must be processed by the *SideBySide* image plug-in. This plug-in adapts the content by merging the reference image and disparity map with a resolution of 960×540 into one single 1080p video stream compliant with the format supported by the Philips Wow display, (See Figure 1).

5. CONCLUSIONS AND FUTURE WORK

We have proposed an approach that achieves real-time performance in a 3D videoconferencing application using autosterescopic displays. We have implemented and optimized several algorithms on GPU trying to find the best compromise between quality of visualization and computational cost. Our approach runs at 25 fps at SD image resolution on an usual PC, with commertial hardware (dual core Intel processor with a NVIDIA GTX 295 GPU), using a multimedia architecture based on GStreamer. We achieve this performance by implementing an optimization of the Adaptive support-weight algorithm. This optimization is able to calculate disparity maps accurately in a short time, with low computational consumption. This efficiency allows us to add some other optimizations such as a background subtraction, cross-check consistency testing or error correction mechanisms such as gap-filling with Anisotropic diffusion. All of these developments have been successfully implemented in GPU using CUDA in order to obtain the highest levels of parallelization and performance possible.

We are currently integrating a Time of Flight (ToF) camera into our architecture. In the future we plan to use this type of camera in combination with digital camera pairs. This combination will result in an hybrid approach that could overcome the problems of each technology when working independently. With this approach we hope to obtain better disparity maps at higher resolutions.

6. ACKNOWLEDGEMENTS

This work has been partially supported by the Spanish Administration agency CDTI, under project CENIT-VISION 2007-1007.

7. REFERENCES

[1] Milton Chen, "Leveraging the asymmetric sensitivity of eye contact for videoconference," in *CHI '02: Proceedings of the SIGCHI conference on Human factors in computing systems*, New York, NY, USA, 2002, pp. 49–56, ACM.

[2] Tam T, Cafazzo JA, Seto E, Salenieks ME, and Rossos PG., "Perception of eye contact in video teleconsultation," *Journal of telemedicine and telecare*, vol. 13, no. 1, pp. 9–35, February 2007.

[3] http://www.3dpresence.eu, "3dpresence," .

[4] A. Vetro, S Yea, and A. Smolic, "Towards a 3d video format for auto-stereoscopic displays," *Proc. Conference on Applications of Digital Image Processing*, vol. 7073, 2008.

[5] Daniel Scharstein and Richard Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[6] Kuk-Jin Yoon and In So Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650, 2006.

[7] Thanarat Horprasert, David Harwood, and Larry S. Davis, "A statistical approach for real-time robust background subtraction and shadow detection," in *ICCV Frame-Rate WS*, 1999.

[8] Pietro Perona and Jitendra Malik, "Scale-space and edge detection using anisotropic diffusion," Tech. Rep. UCB/CSD-88-483, EECS Department, University of California, Berkeley, december 1988.

[9] http://www.gstreamer.net, "Gstreamer," .