Real-time depth map manipulation for 3D visualization

Article *in* Proceedings of SPIE - The International Society for Optical Engineering · February 2009

DOI: 10.1117/12.812405

CITATIONS
2

READS
244

3 authors:

Ianir Ideses

27 PUBLICATIONS **436** CITATIONS

SEE PROFILE

Barak Fishbain

Technion – Israel Institute of Technology

104 PUBLICATIONS **2,826** CITATIONS

SEE PROFILE

Leonid P. Yaroslavsky

Tel Aviv University

257 PUBLICATIONS **3,912** CITATIONS

SEE PROFILE

Real-Time Depth Map Manipulation for 3D Visualization

Ianir Ideses[a*], Barak Fishbain[b], Leonid Yaroslavsky[a]
[a] Faculty of Engineering, Tel-Aviv University, Tel Aviv 69978, Israel;
[b] Department of Industrial Engineering and Operational Research, University of California at Berkeley, Berkeley, CA, USA.

ABSTRACT

One of the key aspects of 3D visualization is computation of depth maps. Depth maps enables synthesis of 3D video from 2D video and use of multi-view displays. Depth maps can be acquired in several ways. One method is to measure the real 3D properties of the scene objects. Other methods rely on using two cameras and computing the correspondence for each pixel. Once a depth map is acquired for every frame, it can be used to construct its artificial stereo pair.

There are many known methods for computing the optical flow between adjacent video frames. The drawback of these methods is that they require extensive computation power and are not very well suited to high quality real-time 3D rendering. One efficient method for computing depth maps is extraction of motion vector information from standard video encoders. In this paper we present methods to improve the 3D visualization quality acquired from compression CODECS by spatial/temporal and logical operations and manipulations.

We show how an efficient real time implementation of spatial-temporal local order statistics such as median and local adaptive filtering in 3D-DCT domain can substantially improve the quality of depth maps and consequently 3D video while retaining real-time rendering.

Real-time performance is achived by utilizing multi-core technology using standard parallelization algorithms and libraries (OpenMP, IPP).

Keywords: Real-Time, 3D, Depth-Maps, 3D-DCT

INTRODUCTION

3D visualization is a growing field that has gained much attention in recent years. Developments in this field include improved 3D visualization devices, namely autostereoscopic and multi view autostereoscopic displays, improved stereo acquisition systems, methods to compute depth maps and rendering algorithms.

One of the key aspects of 3D visualization and, in particular, 3DTV (3D Television) is computation of depth maps. This aspect enables synthesis of 3D video from 2D video and use of multi-view displays. Depth maps can be acquired in several ways, one method is to measure the real 3D properties of the scene objects, using, for example, a range finder, eg. a LADAR (laser RADAR) system. Other methods rely on using 2 cameras and computing the correspondence for each stereo pair pixel. Once a depth map is acquired for every frame, it can be used to construct its artificial stereo pair.

Using depth maps for 3D visualization is superior to acquiring a stereo video stream in that that it allows transformations on the depth and manipulations on the parallax baseline. There are many methods to compute depth maps, among them are the works of Lucas and Kanade [1], Horn and Schunck [2] Senthil Periaswamy and Hany Farid [3], Yu-Te Wu et al [4], L. Alvarez et al [5], Jochen Schmidt [6] and Adee Ran and Nir Sochen [7]. The drawback of these methods is that they require high levels of computation power and are not very well suited to high quality real-time 3D rendering.

A lower computational cost method for computing depth maps is extraction of motion vector information from standard video encoders [8]. In this paper we present methods to improve the 3D visualization quality acquired from compression CODECS by spatial/temporal and logical operations and manipulations in real time.

* ianir@eng.tau.ac.il

Real-Time Image and Video Processing 2009, edited by Nasser Kehtarnavaz, Matthias F. Carlsohn, Proc. of SPIE-IS&T Electronic Imaging, SPIE Vol. 7244, 72440J · © 2009 SPIE-IS&T · CCC code: 0277-786X/09/$18 · doi: 10.1117/12.812405

MOTION FIELD EXTRACTION

MPEG 4 (H.264) is a modern compression standard that uses both temporal and spatial compression. While spatial compression is basically a form of JPEG compression, it is temporal compression that enables the high compression rates of MPEG 4. In temporal compression, each frame is divided into blocks and block search is performed between adjacent frames for the location of the blocks. In this fashion, it is necessary to store the movement of the block from one frame to the other, thus reducing the amount of information to store.

MPEG 4 enables computation of motion vectors in blocks as small as 4X4 pixels with quarter pixel accuracy. These data are very useful for depth estimation. In it's simplest form, the horizontal, X-axis, motion vectors can be used as depth data. This holds for cases where there is only lateral motion on the X-axis and no scene motion is present (a canonical stereo setup). For other motion types it is necessary to make a transformation from motion vector maps to depth maps. In our implementation, motion vector maps were extracted as a part of the MPEG 4 encoder schema. The encoder was instructed to extract motion vectors for every frame (regardless of the ultimate frame type) with the minimal block size.

In some cases, motion vector maps can be directly treated as depth maps. This approximation holds when the two images/frames are taken in parallel viewing or when they are acquired in small ranges of disparity in the case of epipolar acquisition. This is usually the case in computation of depth maps of 2D video. However, there are many cases where this approximation does not hold. This happens when the motion is either too rapid in terms of camera rotation or in the case of camera zoom. Such cases can be detected by analysis of the motion vector maps and dealt with.
In the case of zoom, it is necessary to change the dynamic range of the depth values (while cropping out border pixels), the amount of dynamic range scaling has to be congruent with the zoom factor, for the purpose of visualization this has to be visually comfortable rather than true-to-life accurate. In the case of rotation around a specific object, one needs to invert the disparity values, so that the close object receives high disparity values although it appears to be static. Other depth values should remain the same. This is a non-trivial case and indeed is error prone.
In our implementation, we treated motion as a sole depth cue, namely, we calculated the depth solely on the values of the X and Y motion vector values. The depth was estimated by:

$$D(i,j) = c\sqrt{MV(i,j)_x^2 + MV(i,j)_y^2} \qquad (1)$$

Where $D(i,j)$ is the depth value for pixel (i,j) and $MV(i,j)_x$, $MV(i,j)_y$ are the X and Y motion vectors values for that pixel respectively and 'c' is a custom defined scale parameter.

The scaling parameter 'c' can be utilized in two ways, either automatically, or set as a user selected factor. In our implementation, both methods are supported. One may opt to scale the parameter to fit the maximal disparity over all frames – simply adding a constant gain to the depth map values, or perform automatic scaling unto some predefined parallax, keeping maximal parallax constant in all frames. In essence this operation stretches the dynamic range of all depth maps to this level (a nominal parallax value for comfortable viewing is of the order of 20 pixels).

An example of a resulting depth map extracted from MPEG 4 can be seen in Figure 1. As can be seen in the figure, the resulting depth maps retains the appropriate shape information. However it is noisy and requires filtering for good visualization. In order to improve results and to decrease the noise levels in the resulting depth maps other methods were used – spatial and temporal filtering.

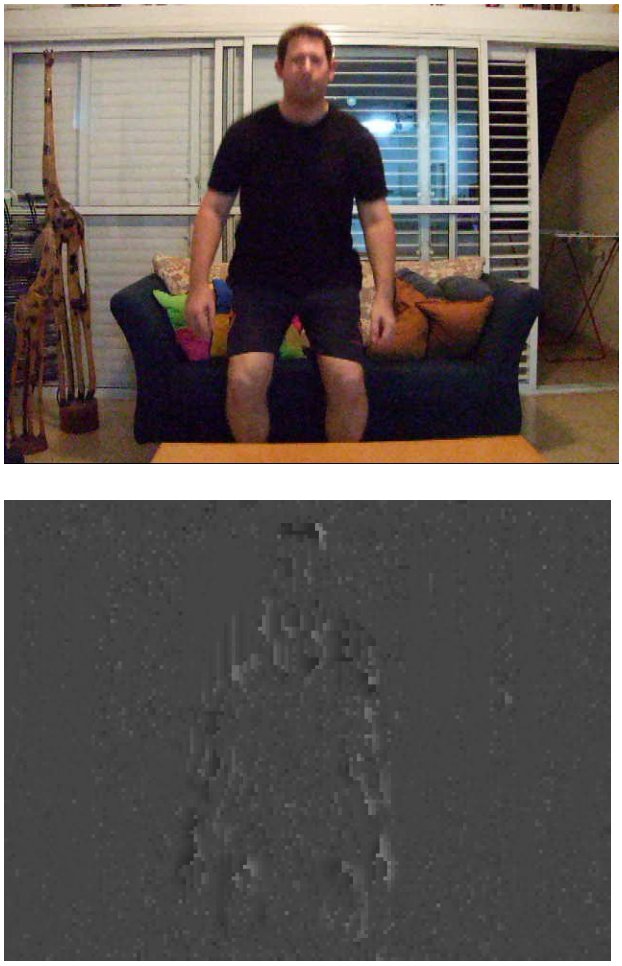


Figure 1. A video frame (top image) and the corresponding depth map (bottom image). Although the resulting depth map does not show the exact metrics of the stereo pair, it is sufficient for the purpose of visualization.

MOTION FIELD EXTRACTION

In order to improve the visual quality of depth maps, and consequentially the resulting 3D images, we propose several methods. Among these are spatial and temporal rank filtering, 3D-DCT and other morphological/logical operations (registration, depth map repetitions, etc').

Analysis of the resulting depth map (sequences can be found in [9]) has shown that these images suffer from erratic behavior in areas of zero motion and in problems associated with global motion and sign inversion when objects change their direction.

The first step in improving these depth maps was to find the global motion value and adjust all other motion vectors to it. This was achieved by computing the histogram of the depth map and finding the highest peak. Sequences that contain global motion exhibit it's value in this peak. In sequences without global motion this value was useful for determining the value for direction inversion (The assumption is that in stationary images, the majority of the motion pixels should be 0). Once this value has been computed, it is subtracted from the entire depth maps image and the absolute value is taken.

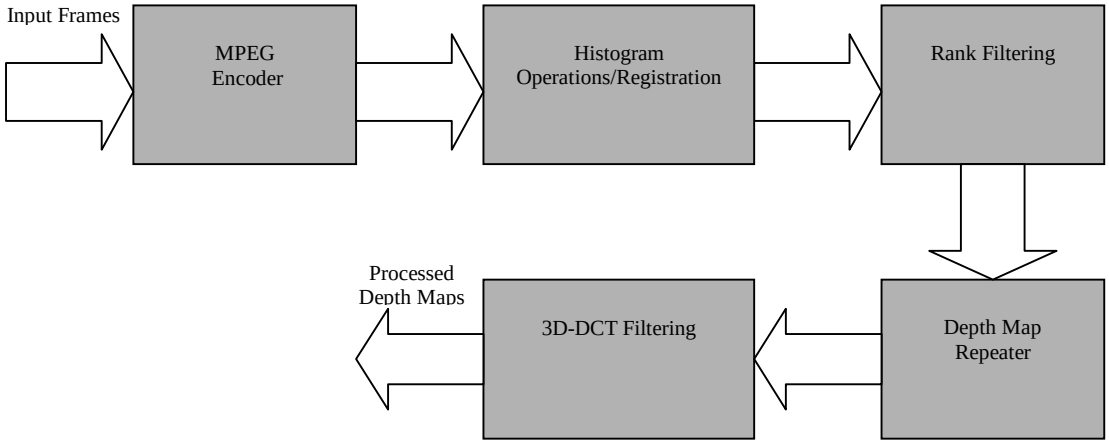
Once the depth map has been normalized it is subjected to spatial-temporal filtering – we used 2 types of filtering, low pass filtering using 3D-DCT and rank filtering. Of these 2 methods, rank filtering proved more useful for the general case, spatial-temporal filtering was achieved in 2 stages, computing the median for each pixel along the time domain, for

global motion scenarios this required prior registration of the images. Computation of the registration parameters was not necessary as it was extracted from the motion histogram. Once images were registered and each pixel's value was computed through a median filter, temporal filtering along a spatial neighborhood was performed. In principle the median operation can be computed over the spatial-temporal cube, but for the sake of computational complexity it was realized separably. This enabled efficient top level parallelization as described in the next sections.

Selecting the neighborhood for temporal filter was based on evaluation of the motion field, we chose a value that enabled a reasonable amount of pixels to be evaluated (this becomes important mostly for global motion scenarios). The parameter that we chose was 7 frames. This amount to about ¼ of a second for normal video standards and enables good filtration without artifacts due to major scene changes.

Selecting the Spatial neighborhood relies on previous results attained for depth map resolution and quantization [10-11]. These results indicate that the depth map resolution can be at least one order of magnitude lower than that of the image itself. Therefore we chose a large filter that corresponds to 1/8 of the image size on each axis.
Due to the nature of the method of depth map computation (shape from motion). It may happen that the depth maps be completely zero. This can happen when no motion is present, even though the shape was previously extracted. In these cases we use the same depth map again until motion is detected again.
3D-DCT is an extension of conventional 2D-DCT to the temporal domain [12 Alex paper/thesis ?] in this case one can design filters that would act as noise suppressors/smoothers in both the spatial and temporal domains. These filters exhibit superior performance to standard 2D filters [12].
In our implementation, we used 3D-DCT filtering as the last stage, filtering was implemented as a cube of spatial LPF filters whose maximal point decreased along the temporal domain. In most cases this filtering was not necessary due to the good noise suppression capabilities of the rank filtering.

A flow chart of the process is shown in figure 2.



S3D RENDERING

Once the depth maps are computed for each frame, a stereo pair can be rendered. Rendering is performed by resampling the incoming frame from the bitstream on a grid that is governed by the depth map. This process is described in equation 2.

$$P_R(i,j) = P_L(i, j + MF * D(i,j)) \tag{2}$$

where $P_R(i,j)$ is the pixel value of the rendered right image in location (i,j), $P_L(i,j)$ is the initial left image value at (i,j), MF is the magnification factor, in our case 16 and $D(i,j)$ is the depth value at (i,j).

This process creates a synthetic view that corresponds to the right stereo pair image. Once this image is computed it can be used to synthesize 3D frames.

For anaglyph viewing, it is sufficient to render only one color component, but is not so for other visualization devices. Autostereoscopic displays, as well as shutter glasses and polarized glasses usually require 2 full color images for 3D visualization (it has been shown that 3D visualization can also be performed using 1 color image and 1 grayscale image [13], but in some cases, it is required that both images be in color). To this end, it is required to perform this operation threefold. This can be achieved efficiently by dividing this operation into threads that retain the same depth map data and reduce memory overheads.

3D RESULTS

Once we rendered a stereo pair for each frame, we could use standard 3D visualization techniques. The most suited to standard display hardware is anaglyphs. A resulting depth map and anaglyph can be seen in figure 3.

As can be seen, we were able to filter out the noise that is present in this type of motion field based depth maps and attain very good visualization. In addition, further processing of the stereo pair (smoothing and dynamic range manipulation are also possible [13]). Additional frames and video sequences are available online [9]

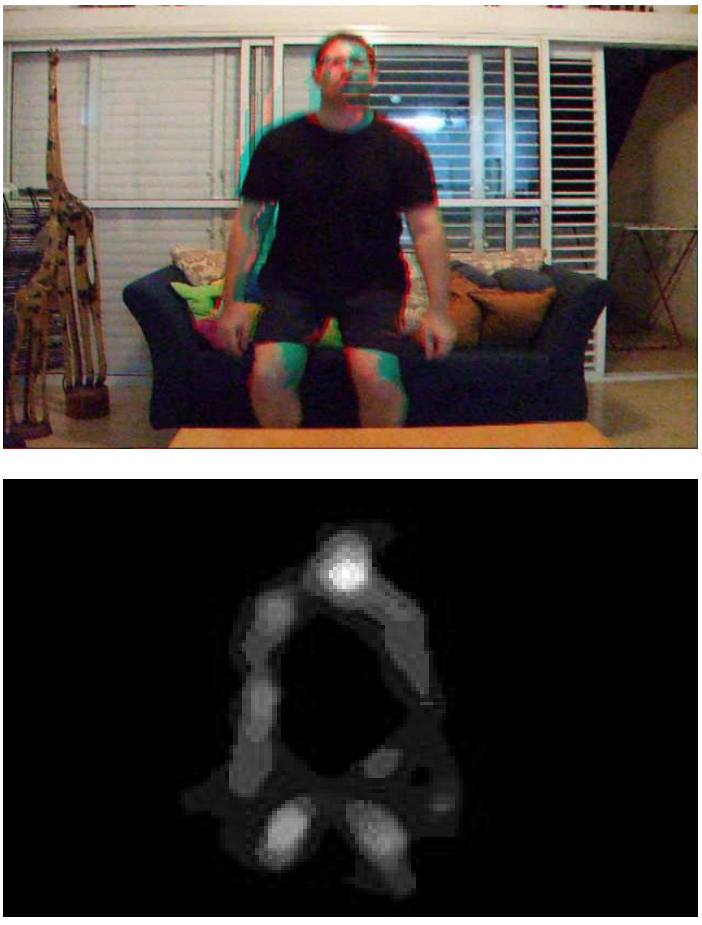


Figure 3. Top image - an example of a 3D image (image viewed with anaglyph glasses, red filter for right eye, blue filter for left eye). Bottom image – corresponding depth map

.

REAL-TIME CONSIDERATIONS

Synthesizing artificial 3D video, and especially conversion of 2D video to 3D are very computationally intense operations. These include decoding of the input video, computation of depth maps, rendering of artificial stereo pairs and combining the stereo pair to a 3D image.

In order to enable real time performance, we chose two methodologies:

a. Efficient micro code execution and code vectorization.
b. Efficient multi-threading for multi-core platforms.

The first methodology was implemented via Intel Integrated Performance Primitives (IPP), a standard library that enables usage of readily available algorithms that were optimized for Intel microprocessors and is thread optimized.

The second methodology was implemented via OpenMP, a standard library that enables simple, yet efficient code parallelization without significant code rewrite.

The task at hand – real time conversion and depth map manipulations involves several cascading stages. In principle, one could argue that since the output is video frames, it is possible to parallelize the cascade for each frame, so that each processor core handles one frame. The reality is that due to the stage of temporal filtering, simple frame parallelization would be very inefficient, requiring to recompute DCT and rank filters for the same frames several times.
In order to overcome this obstacle, we implemented 3D-DCT in a running temporal window and computed the rank filters separably. Parallelization of these algorithms was carried out inside the blocks, thus employing all available processor cores at all cascades. Utilizing the OpenMP pragma directives enabled simple design. IPP libraries were used whenever possible to further utilize multi-thread operations.

A diagram of the multi threading design is shown in Figure 4:

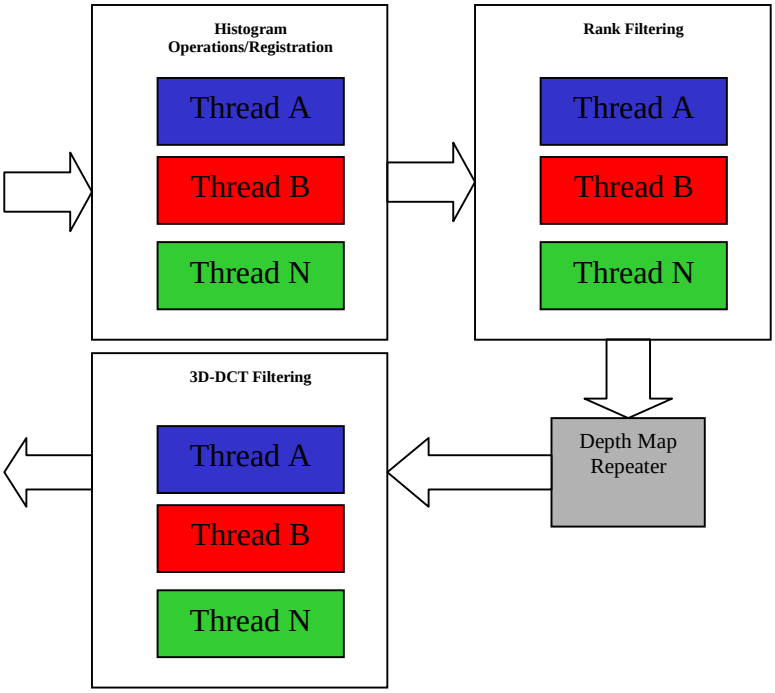


Figure 4: Parallelization of the depth map manipulation stage, as can be seen, parallelization was performed at the cascade element stage. Emphasis was given to computationally intense operations, namely: 3D-DCT, rank filtering and registration. Simpler algorithms, such as depth map repetition require very little computational load and are not good candidates for parallelization.

Testing of this parallelization was performed by using operating system tools for CPU load measurements. These have shown almost full utilization of all system cores. Such load experiments were conducted on quad core systems as, well as 8 core systems with similar results.

The system was implementd in the Matlab development environment.Computationally intense operations where Matlab does not exhibit good performance were coded in C++ using the stated above methodologies (IPP, OpenMP) and compiled as MEX functions. This enabled simple system development and performance profiling.

CONCLUSION

In this paper we have described methods to manipulate depth maps for the purpose of 3D visualization. We have shown that even noisy maps that can be extracted from motion compensation based COEDCS can be filtered to achieve good 3D visualization while retaining real-time performance.

Synthesizing artificial 3D video, and especially conversion of 2D video to 3D are very computationally intense operations. These include decoding of the input video, computation of depth maps, rendering of artificial stereo pairs and combining the stereo pair to a 3D image. In order to achieve real-time performance we chose 2 approaches. The first is utilization of information stored in the compressed video stream, namely, motion vector information. The second is efficient implementation of multi-threaded code on multi-core platforms.

We used state of the art standard libraries that enable vectorization and efficient processor specific micro code such as IPP and powerful algorithmic solutions for thread parallelization (OpenMP). Using these methodologies, we were able to utilize multi-core processors and ensure that all cores are utilized to the maximum, thus reducing CPU idle cycles and achieving real-time performance.

REFERENCES

[1] Lucas, B., and Kanade, T. “An Iterative Image Registration Technique with an Application to Stereo Vision”, in Proceedings of 7th International Joint Conference on Artificial Intelligence (IJCAI), pp. 674-679, 1981.

[2] B. Horn and B. Schunck, “Determining Optical Flow”, Artificial Intelligence, 17:185-203, 1981.

[3] Senthil Periaswamy, Hany Farid ", Elastic Registration in the Presence of Intensity Variations", IEEE Transactions on Medical Imaging, July 2003, Volume 22, Number 7

[4] Yu-Te Wu, Takeo Kanade, Ching-Chung Li and Jeffrey Cohn, Image Registration Using Wavelet-Based Motion Model, International Journal of Computer Vision, July 2000.

[5] L. Alvarez, R. Deriche, J. Sanchez, and J. Weickert. Dense Disparity Map Estimation Respecting Image Discontinuities: A PDE and Scalespace Based Approach. Technical Report RR-3874, INRIA, January 2000.

[6] Jochen Schmidt, Heinrich Niemann, and Sebastian Vogt. Dense disparity maps in real-time with an application to augmented reality. Orlando, FL USA, December 3-4 2002. IEEE Computer Society.

[7] Adee Ran, Nir A. Sochen: Differential Geometry Techniques in Stereo Vision. 98-103, EWCG 2000 Eilat Israel.

[8] Ianir Ideses, Leonid P. Yaroslavsky and Barak Fishbain. Real-Time 2D to 3D Video Conversion. Journal of Real-Time Image Processing, Volume 2, Number 1 / October, 2007.

[9] http://www.eng.tau.ac.il/~ianir

[10] I.Ideses, L.P.Yaroslavsky, B.Fishbain, How Sharp Must Depth Maps Be for Good 3-D Video Synthesis: Experimental Evaluation and Applications, ICO Topical Meeting on Digital Holography and Three-Dimensional Imaging, 2007, Vancouver Canada.

[11] I.Ideses, L.P.Yaroslavsky, I.Amit and B.Fishbain, Depth Map Quantization – How Much Is Sufficient ?, 3DTV CON, Kos, Greece 2007.

[12] L. P. Yaroslavsky, Digital Holography and Digital Image Processing (Kluwer Academic, Boston, Mass., 2004), pp. 199–209.

[13] I.Ideses and L.P.Yaroslavsky, Three Methods that improve the visual quality of colour anaglyphs, 2005 J. Opt. A: Pure Appl. Opt. 7 755-762.