Implementation of a Dynamic Image-Based Rendering System

Niklas Bakos[1], Claes Järvman[2] and Mark Ollila[3]

Norrköping Visualization and Interaction Studio
Linköping University

Abstract

Work in dynamic image based rendering has been presented by Kanade et al. [4] and Matusik et al. [5] previously. We present an alternative implementation that allows us to have a very inexpensive process of creating dynamic image-based renderings of digitally recorded photo realistic, real-life objects. Together with computer vision algorithms, the image-based objects are visualized using the *Relief Texture Mapping* algorithm presented by Oliveira et al [6]. As the relief engine requires depth information for all Texels representing the recorded object in an arbitrary view, a recording solution making depth extraction possible is required. Our eyes use binocular vision to produce disparities in depth, which also is the most effortless technique of producing stereovision. By using two digital video cameras, the dynamic object is recorded in stereo in different views to cover its whole volume. As the depth information from all views are generated, the different views from the image-based object are textured on a pre-defined bounding box and relief textured into a three dimensional representation by applying the known depth disparities.

1 System Prototype

The first step in the process is to record a dynamic object in stereo, which gives us the photo textures for the image-based object and the possibility to derive depth information from the stereo image-pairs. To be able to use the recorded video as a texture when rendering, it is important that one camera (i.e. the left) is installed parallel to the normal of the sides of the bounding box surrounding the object, and the other (i.e. the right) next to, in a circular path so that both cameras have the same radius to the object. As we are interested in the recorded object only, the image background should be as simple as possible. By using a blue or green screen, the object can easily be extracted later on. A blue screen can easily be installed by using cheap blue matte fabric on the walls. Depending on the amount of cameras available, the dynamic object is recorded in stereo in up to five views (front, back, left, right and top). In this project, only two cameras were used, giving us only one view when filming the dynamic object. As the recording is finished, the video streams are sent via firewire to a PC, where the resolution is rescaled to 256x256 pixels, the background is removed and the depth maps are calculated, enhanced, cropped and sent to the relief rendering engine. (Pipeline in figure 1).

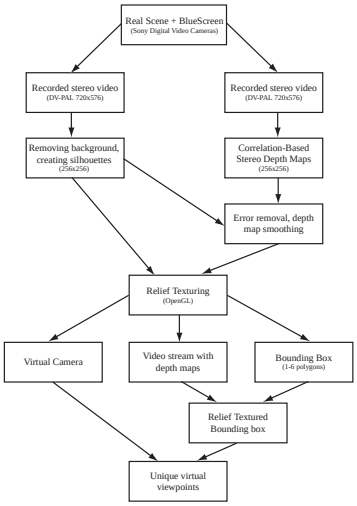


Figure 1: Prototype overview. *A schematic view over the different stages required in the process of rendering new views of an image-based object.*

2 Depth approximation

When the stereo video have been recorded and streamed to the computer client, our algorithms start processing the data to create useful video frames and information about the scene. As the objects are extracted from the original video, the process of estimating the depth of the scene is initiated. When the approximated depth map for a certain frame is generated, it is used together with the object image to render unique views, using the relief-rendering engine. This session starts with a brief overview of the depth algorithm, followed by complete descriptions about all the steps from using original video streams to sending a finalized depth map and video frame to the rendering process of virtually viewing the object from an arbitrary view.

[1] nikba@itn.liu.se
[2] claja622@student.liu.se
[3] marol@itn.liu.se

2.1 Algorithm overview

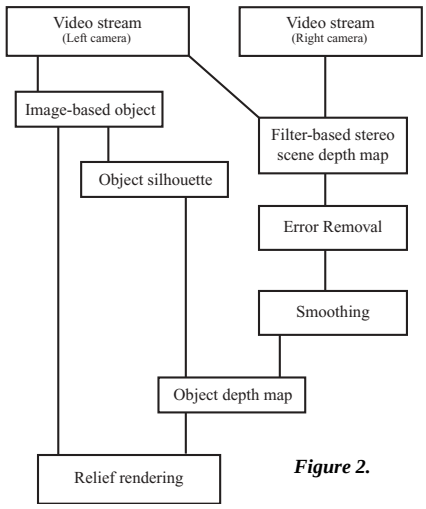


Figure 2.

A summary of the algorithm pipeline is shown in figure 2. From the *N* stereo video cameras, we have *2N* video streams. From the left camera (which sees the scene straight from the front), the object-only video frames and silhouette will be created. As the scene is recorded with a blue screen background, both the silhouette and the object extraction are created rapidly. Simultaneously, both the left and the right video streams are segmented into frames and sent into our filter-based depth algorithm. At this stage, the frames can be downsized for optimization purpose, which will result in faster depth map approximations with lower quality. For each frame, each pixel from the left image is analyzed and compared with a certain area of the right image to find the pixel correspondence. With this known, the depth could be estimated for each frame. Since this mathematical method outputs a relatively distorted image, it needs to be retouched to fit the relief engine better. First, the depth map is sent to an algorithm for detecting edges, where an edge could be thought of as noise, distorting the depth map, and removed by pasting the intensity value of neighboring pixels. With the errors removed, the depth approximation of the image-based object will contain less noise and unnecessary holes, but disparities between contiguous object regions might be rendered with too sharp intensity variances, which will exaggerate the displacement of some object parts when applying the relief mapping. To solve this, the depth map is smoothened and finally, the silhouette is added to remove approximated background depth elements.

2.1.1 Filter-based stereo correspondence

The method implemented in our system prototype uses filter-based stereo correspondence developed by *Jones* and *Malik* [2], a technique using a set of linear filters tuned in different rotations and scales to enhance the features of the input image-pair for better correlation opportunities. A benefit of using spatial filters is that they preserve the information between the edges inside an image. The bank of filters is convolved with the left and the right image to create a response vector at a given point that characterizes the local structure of the image patch. Using this information, the correspondence problem can be solved by searching for pixels in the other image where the response vector is maximally similar. The reason for using a set of linear filters at various orientations is to obtain rich and highly specific image features suitable for stereo matching, with fewer chances of running into false matches. The set of filters F_i (fig. 3) used to create the depth map consists of rotated copies of filters generated by

$$G^{\chi}_{n,0}(x,y) = G_n(u)\Delta G_0(v) \;\; ; \;\; u = x\cos\chi - y\sin\chi \,, \;\; v = x\sin\chi - y\cos\chi$$

where *n=1, 2, 3* and G_n is the n^{th} derivative of the Gaussian function, defined as

$$G_0(x) = \frac{1}{\sqrt{2\phi\omega^2}} e^{\frac{-z^2}{2}} \;\; ; \;\; z = \frac{x}{\omega} \quad G_1(x) = -\frac{1}{\omega} z G_0 \;\; ;$$

$$G_2(x) = \frac{1}{\omega^2}(z^2 - 1)G_0 \;\; ; \;\; G_3(x) = -\frac{1}{\omega^3}(z^3 - 3z)G_0 \,.$$

The matching process was performed using different filter sizes to find the optimized filter settings, resulting in an *11x11*-sized matrix with a standard deviation value ω of *2*. The number of filters used depends on the required output quality. Using all filters would result in a high detailed depth approximation, but the processing time would be immense. Testing different filters to optimize speed and output quality, the resulting filters consisted of nine linear filters at equal scale, with some of them rotated, as shown below.

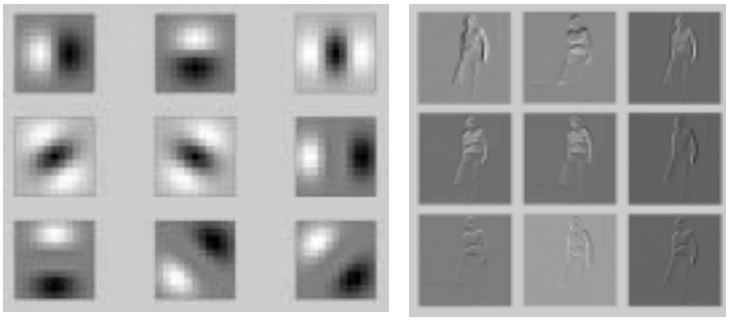


Figure 3: Spatial filter bank. *Image plots of the nine filters generated by copies of rotations of Gaussians.*

Figure 4: Response vectors. *An illustration of how the response vectors will look like after being convolved with different filters. In reality, a response vector never represents a whole image.*

The disadvantage of using one scaling level only is the loss of accuracy when matching pixels near object boundaries or at occluded regions. But again, using more scales, the rendering time will increase proportionally. To search for pixel correspondence, an iterative process is created, scanning the left image horizontally, pixel by pixel, left to right, and seeks for similar intensity values inside a defined region surrounding the current pixel location. For each row, the set of linear filters are convolved with a region of the right image determined by its width and the height of the filter size, to create a response vector that characterizes the features of this row. At this row, a new response vector for each pixel is created by convolving the filter bank with a filter-sized region from the left image. How the convolved response vectors for a whole image would look like is illustrated in figure 4. (Note that the response vectors are only representing a small region of the image for each iteration of the correspondence process).

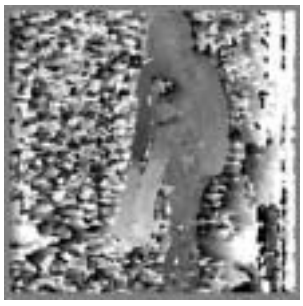
$$v_{i,right} = \text{Right image } (r)\ 1\ F_i = \sum_{x'\ y'} r(x', y')\, F_i(4x', y'4y')$$

$$v_{i,left} = \text{Left image } (l)\ 1\ F_i = \sum_{x'\ y'} l(x', y')\, F_i(4x', y'4y')$$

The convolving returns only those parts of the convolution that are computed without the zero-padded edges, which minimizes the response vectors and optimizes the whole process of finding the correspondence. As soon as the images are convolved with the filters, the matching process for finding the correlation is initiated. To restrict the searching area, a one-dimensional region needs to be determined. By using a small region, the corresponding pixels may not be found, as the equivalent pixel probably is located outside this region. On the other hand, if the region is too large, a pixel not related to that area might be thought of as correct. When the region is established, this is used to crop the response vector $v_{i,right}$ created from the right image. When the response vectors are defined at a given point, they need to be compared in some way to be able to extract some information about how the pixels are related. By calculating the length of their vector difference *e*, which will equal zero if the response vectors are identical, this can be used to solve the correspondence problem. This is done by taking the sum of the squared differences (SSD) of the response vectors,

$$e = \sum_{i} (v_{i,left} 4 v_{i,right})^2$$

Figure 5.



where *i* is the amount of filters used and the pixel position (defined as *k*) containing the value closest to zero is saved. When the correspondence has been established, the disparity has to be defined to be able to create a depth map. For each pixel in the left image, we know the position of the matching pixel in the right image. To create a connection between this data, the depth value $d(i, j)$ for each pixel could be estimated by

$$d(i, j) = k 4 i$$

where *k* is the horizontal position of the corresponding pixel and *i* is the current pixel position. The depth map (fig. 5) is approximated with intensity levels depending on the size of the constant defining the size

of the matching region and if a corresponding pixel is found to the left of current pixel *i*, the intensity is set to a value pointed to white, and vice versa, depending on the rotation of the image-pair.

2.1.2 Locating errors and noise

The primary depth map image generated by the filter-based stereo algorithm is a general approximation of the depth information regarding the objects in the video frames. As this algorithm has no knowledge in form of estimating the structure of object connectivity or how the scene is designed, unpredicted outputs might appear. They can be found by convolving the image with an edge detection filter [7]. The operator best suited for our needs turned out to be the *Robinson* filters h_1 and h_3.

$$h_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & -2 & 1 \\ -1 & -1 & -1 \end{pmatrix}$$

$$h_3 = \begin{pmatrix} -1 & 1 & 1 \\ -1 & -2 & 1 \\ -1 & 1 & 1 \end{pmatrix}$$

Figure 6.

With the vertical and the horizontal Robinson filters defined, they are convolved with the depth map to find obvious edges in it, using the convolution formula for two dimensions. We now have two temporary depth map images, with the edges defined vertically and horizontally, shown in figure 6. From this, the edge magnitude of each pixel could be derived as

$$d(x,y) = |d_1(x,y)| + |d_2(x,y)| = \sqrt{d_1^2(x,y) + d_2^2(x,y)}$$

The result is shown in figure 7a and gives a better analysis of how the errors are structured. To be able to use this information cleverly, the pixels convolved and defined as positions of eventual errors need to be saved. Also, these pixels need to be easily accessed. By using a threshold value, we can decide which of the convolved 'edge'-pixels that will belong to the error pixels in the original depth map, shown in figure 7b. With the positions of the erroneous pixels known, they are replaced by neighboring pixel values, which creates a smoother depth map, although not mathematically perfect, since it is only assumed that these pixels have the same properties as the invalid and replaced ones. On the other hand, the noiseless depth maps, shown in figure 8, will generate tremendously enhanced renderings when applied by the relief engine.

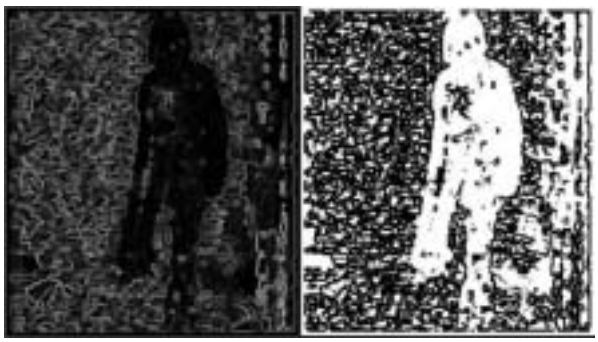


Figure 7a & b.

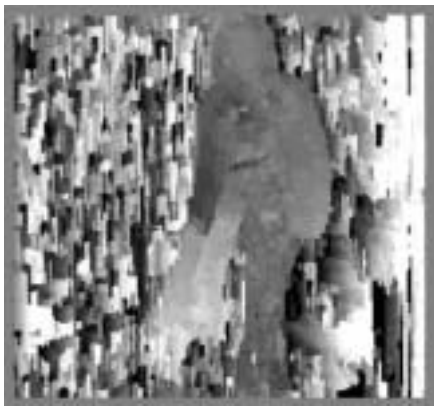


Figure 8.

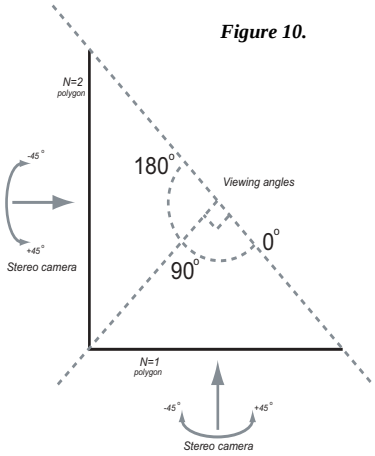
Figure 9.

2.1.3 Smoothing the depth map

The output from the edge detection process is a more or less error free depth map, regarding the hole filling and the depth intensity interpretation. On the subject of intensity, it can fluctuate significantly over connected and contiguous surfaces over the object. As some intensity values diverges in areas were they actually would be similar, the solution would be to decrease the higher values and increase the lower to create more similar intensities over that specific area, in other words, smoothing the image. This might generate an intensity value incorrect for the true depth of that part of the object, but applying this solution to the whole image, the displacement would act as an intensity threshold only. The Gauss function is used to generate a smooth depth map, defined as the well-known *Gaussian blur* filter [1]. We defined a Gaussian operator and convolved it with the depth map to obtain the smooth result, seen in figure 9.

2.1.4 Rendering

Figure 10.



A fully functional application for the relief rendering of the image-based object and its depth maps was written in C++ using OpenGL, created in parallel to this project [3] and modified to fulfill the criterion of our system prototype. The number of polygons required for rendering equals the amount of stereo cameras used. Because of the good depth information approximated with the filter-based stereo algorithm, the viewing angle was set to $\partial 45N^{\circ}$ from the center of the origin of the textured polygon box, illustrated in figure 10.

3 Results

The resulting application consists of two demos (screenshots available on the last page):

- Static demo (yellow pullover) - Requires two input textures and with two depth maps, textured on two polygons. From two original views, with 90 degrees separation, new unique views can be created within 180 degrees. The polygons are mapped with textures of size 256x256 pixels and the frame rate is ~15 frames/sec.
- Dynamic demo (pink pullover) - Representing a person walking around. Textured on only one polygon, which restricts the viewing angle to 90 degrees. The amount of input data required depends on the frame rate. We used a frame rate of 20 frames/sec, with a video buffer of 40 images and 40 depth maps. The relief engine had no problems with rendering a constantly updating image buffer and the animated sequence showed no indications of flickering.

References

[1] BOGACHEV, V. 1998. Guassian measures. *Mathematical Surveys and Monographs 62*.
[2] JONES, D., AND MALIK, J. 1992. "A computational framework for determining stereo correspondence from a set of linear spatial features". In *EECV*, 395–410.
[3] JÄRVMAN, C., "Static and Dynamic Image-Based Applications using Relief Texture Mapping", Linköping University, LITH-ITN-MT-20-SE. May 2002.
[4] KANADE, T., NARAYAN, P., AND RANDER, P. W. 1997. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia 4*, 1, 34–47.
[5] MATUSIK, W., BUEHLER, C., RASKAR, R., GORTLER, S. J., AND MCMILLAN, L. 2000. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 369–374.
[6] OLIVEIRA, M. M., BISHOP, G., AND MCALLISTER, D. 2000. Relief texture mapping. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 359–368.
[7] SONKA, M., HLAVAC, V., AND BOYLE, R. 1996. *Image Processing, Analysis, and Machine Vision*, second ed. Brooks/Cole Publishing Company.