Real-time Plane-Sweep with local strategy

Vincent Nozick Sylvain Michelin Didier Arquès

SISAR team,
Marne-la-Vallée University, ISIS Laboratory,
6 cours du Danube, France, 77 700 Serris
{vnozick,michelin,arques}@univ-mlv.fr

ABSTRACT

Recent research in computer vision has made significant progress in the reconstruction of depth information from two-dimensional images. A new challenge is to extend these techniques to video images. Given a small set of calibrated video cameras, our goal is to render on-line dynamic scenes in real-time from new viewpoints. This paper presents an image-based rendering system using photogrametric constraints without any knowledge of the geometry of the scene. Our approach follows a plane-sweep algorithm extended by a local dynamic scoring that handles occlusions. In addition, we present an optimization of our method for stereoscopic rendering which computes the second image at low cost. Our method achieves real-time framerate on consumer graphic hardware thanks to fragment shaders.

Keywords

Image-based rendering, plane-sweep, fragment shaders.

1. INTRODUCTION

Given a set of images from different viewpoints of a scene, we set out to create new views of this scene from new viewpoints. This reconstruction problem is treated from several approaches. Some methods focus on the geometry of the scene while others use photogrametric properties. These methods can also differ on the number of input images, on the visual quality of the views created and on computation time. Most of the past work in this field concerns static scenes and tries to improve reconstruction accuracy, but past years, dynamic scene reconstruction has become a more important research area.

Journal of WSCG, ISSN 1213-6972, Vol.14, 2006
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

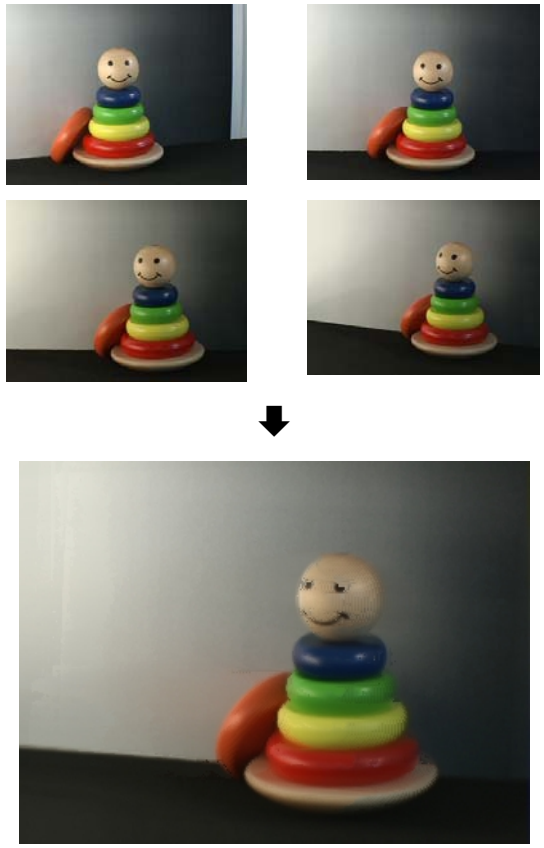


Figure 1 : A real-time reconstruction example from four cameras

In this paper, we present an overview of image-based real-time rendering for static and dynamic scenes. We detail one of these known as the plane-sweep algorithm, and present an adaptation of this method that handles occlusion. Our method achieves real-time framerate on consumer graphic hardware using fragment shaders. We also introduce an computation optimization of our method for stereoscopic rendering.

2. RELATED WORK

This section surveys previous work on real-time image-based rendering (IBR) techniques and real-time reconstruction for static and dynamic scenes.

Real-time rendering

Some image-based methods like the Plenoptic modeling [MB95], the Lumigraph [GGSC96] and the Light Field rendering [LH96] provide real-time photorealistic rendering using a large set of input 2D image samples. Nevertheless these methods require considerable off-line processing before visualization so the handling of dynamic scenes becomes very difficult. Schirmacher et al. [SLS01] extend a Lumigraph structure with per-pixel depth information using a depth-from-stereo algorithm and reach interactive-time at the cost of visual quality.

Depth-from-stereo algorithms [SS02] like SRI SVS [BBH03] provide real-time depth-maps computation from two input video streams without any special purpose hardware. However they do not provide a real-time rendering method synchronized with the real-time depth-map.

Other reconstruction methods such as texture-mapped rendering [PKV00] provide fluid navigation in the reconstructed scene but they require lengthy computation time before visualization.

Dynamic scene rendering

These last methods compute new views of a scene in real-time but most of them begin with a significant preprocessing which prevents them from computing dynamic scenes. In recent years, alternatives to this preprocessing problem and new solutions have been ardently investigated.

A first solution to this problem is to make a preprocessing on a set of videos rather than on a set of images. This allows navigation in dynamic scenes in real-time but these methods can only render playback video. Kanade et al. choose this approach with their Virtualized Reality^{TM} System [KRN97] and achieve real-time rendering with a collection of 51 cameras mounted on a geodesic dome of 5 meters diameter. Zitnick et al. proposed a color-segmentation based stereo algorithm [ZBUWS04] providing high visual quality image in real-time from a set of 8 or more cameras but again, this method involves preprocessing.

Some other real-time techniques handle on-line video flows. Matusik et al. provide an efficient real-time rendering method with their image-based visual hulls [MBRGM00] using a set of four cameras. This method shades visual hulls from silhouette image data but therefore can not handle concave objects.

Finally, some methods like [IHA02] are based on color matching between different views, according to the epipolar constraint. Collins [C96] introduces the plane-sweep algorithm and provides basic reconstruction from binary images. Yang et al. [YWB02] extend this method to color-images and present a real-time implementation using graphic hardware. Woetzel et al. [WJKR04] adapt this method for real-time depth-mapping and introduce a first approach to handling occlusions. Geys et al. [GKV04] use a plane sweep algorithm to generate a crude depth map cleaned up using a graph-cut algorithm.

Our algorithm belongs to the latter family. We will first expose the basic plane-sweep algorithm and [YWB02, WJKR04, GKV04] contribution. Then we will detail our method.

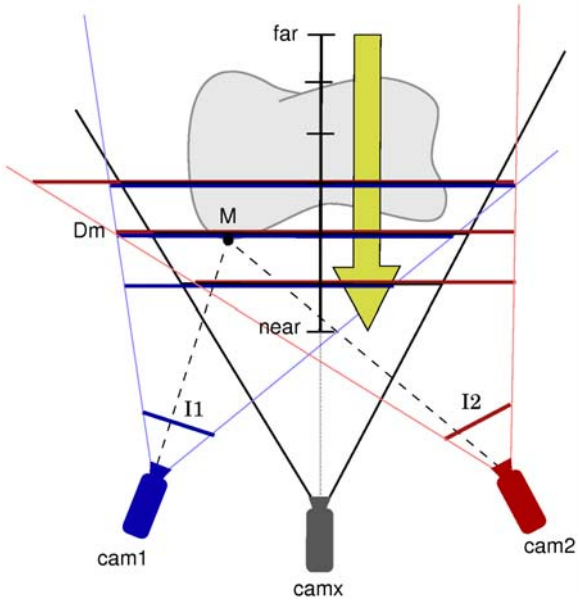


Figure 2 : Plane-sweep algorithm with two input cameras cam_1 and cam_2. M is a point of an object lying on one of the planes D_m in front of the virtual camera cam_x. The input cameras will project M's color on the same pixel of D_m.

3. PLANE-SWEEP ALGORITHM

The initial plane-sweep algorithm was introduced in 1996 by Collins [C96]. He first applied an edge detector filter on the input images and provided a geometric reconstruction of the scene from these binary images. The following overview is an adaptation of this method to color-images.

Overview

Given a small set of calibrated images from video cameras, we wish to generate a new view of the scene from a new viewpoint. Considering a scene where objects are exclusively diffuse, we first place the virtual camera and divide space in parallel planes D_i in front of the camera as shown in Figure 2. We project the input images onto each plane D_i in a back to front order. Let's consider a visible object of the scene lying on one of these planes at a point *M*. The input cameras will project on *M* the same color (i.e. the object color). Therefore, points on the planes D_i where projected colors match together potentially correspond to an object of the scene.

Let I_1 ... I_n denote a set of *n* calibrated images. I_x is the new image to be computed and cam_x is its virtual pinhole projective camera. We define a *near* plane and a *far* plane parallel to cam_x image plane such that all the objects of the scenes lie between *near* and *far*. For each pixel of each plane D_i, a score and a color are computed according to the matching of the colors. The plane-sweep algorithm can be explained as follows :

- initialize I_x's score
- **for** each plane D_i from *far* to *near*
  - → project all the input images I_1...I_n on D_i as textures
  - → project D_i multi-textured on I_x
  - → **for** each pixel *p* of I_x
    - compute a score and a color according to the coherence of the colors from each camera's contribution
    - **if** the score is better than the previous ones **then** update the score and the color of *p*
- draw I_x

Figure 3 shows samples of multitextured planes D_i. When a plane pass through an object of the scene, this object becomes sharp on the multitextured image. This is the case for the wood head on the top right image.

What this method does in effect is comparing epipolar lines between the input images from each pixel of I_x. This method also provides depth-maps by drawing D_i's depth rather than a color.

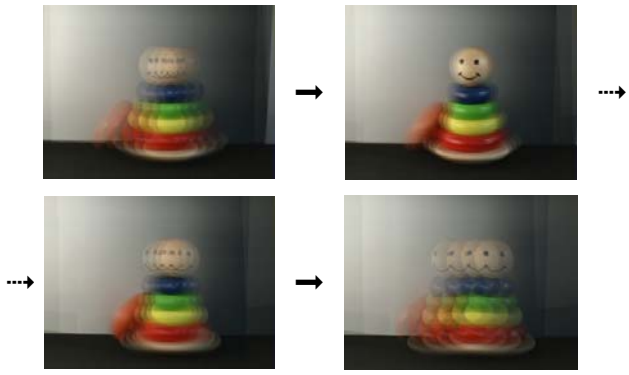


Figure 3 : Pictures associated to four planes D_i using four input images

Like several IBR techniques, this basic algorithm does not handle occlusion since the score is only computed according to the coherence of a small set of colors. We present in section 4 a modification of this algorithm that handles occlusion.

Classical score computation

Yang et al. [YWB02] propose an implementation of the plane-sweep algorithm using register combiners. For the scoring stage, they choose a reference camera cam_{base} that is closest to cam_x and compare the contribution of each input image with the reference image. Each pixel score is computed by adding the Sum of Squared Difference (SSD) from each input images. The SSD (1) compares the luminance of a pixel Y_i of an input image I_k with the corresponding luminance Y_{base} from the reference camera.

$$SSD(Y_i, Y_{base}) = \sum_i \left(Y_i - Y_{base}\right)^2 \qquad (1)$$

For more robustness, they use mipmapping to combine the pixels' score with a score computed from the same images with a lower level of detail. According to the small number of instructions, this method provides good speed results, however the input cameras have to be close to each other and the navigation of the virtual camera should lie between the viewpoints of the input cameras, otherwise the reference camera may not be representative of cam_x. Lastly, there may appear discontinuities in the computed images when the virtual camera moves and changes its reference camera. They propose a register combiners implementation and reach real-time rendering for dynamic scenes using five input cameras.

Woetzel et al. [WJKR04] propose a plane-sweep system that provides real-time depth-maps. Contrary to Yang et al. [YWB02], they do not choose a reference camera but they still compare the input images by pairs. They compute the *SSD* of each pair of input images and sort out the contribution of the

two worse scores. It is a first step to handling occlusions but this method applies the same treatment to each pixel without selecting those which are concerned by occlusion and those which are not. They propose a real-time depth-map method but do not propose any rendering algorithm. This makes the comparison between our algorithms difficult.

The same problem of scoring and choosing colors among a set of colors from epipolar lines has been treated by Fitzigibbon et al. [FWZ03]. They use priors under a large set of input images to choose the color (and hence the depth) that corresponds best to most of the input images. This method is well adapted to a large set of input images and provides good results. However it requires too much computation time for real-time rendering.

Finally, Geys et al. [GKV04] propose a two steps method using two input cameras. First, a plane sweep (GPU) computes a depth map using a Sum of Absolute Differences (SAD) from the two input images. Then, an energy minimisation method (CPU) cleans up the depth map. The energy function considers temporal and spatial continuity, the previous SAD and an occlusion term derived from a background-foreground repartition of the scene elements. The energy function minimisation is solved by a graph cut method and provides a consequent improvement of the initial depth map. View dependent texture mapping of the two input images is performed to create the new view. However, this method requires a background-foreground scene decomposition with a static background. [GV05] introduces an adaptation of this method for three or more cameras.

4. OUR METHOD

We propose a new implementation which makes it possible to take into account all input images together where other methods compute images pair by pair. We introduce new methods using local strategy to compute scores allowing independent treatment of each pixel of I_x in order to handle occlusions. We also propose a new algorithm providing a stereoscopic pair of images with the second view at low cost.

New scores computation

The score computation is a crucial step in the plane-sweep algorithm. Both visual results and speedy computation depend on it. We propose a new method to compute a score according to all the input image colors instead of computing by pairs. For this purpose, we use multi-texturing functions to access each input camera color contribution.

For each pixel of I_x, we propose a finite number of positions X in the scene (one per plane D). A score is computed for each position and this score depends on the color C_i of the projections of X in each input image. We propose three methods to compute scores.

A first possibility is to set the score as the variance of each color C_i and the final color as the average of the C_i. This method is easily implemented and provides good visual results especially if the input cameras are close together. However this method does not handle occlusions. Indeed, a point viewed by all the input cameras except one will have its score and its color distorted since this camera may increase the variance and spoil the average. Nevertheless, this method implicitly treat occlusions when the virtual camera is near from an input camera which projects for each planes D_i approximatively the same image.

We also propose an iterative algorithm to reject outlier colors using a sigma clipping technique. This method first computes the variance v of the color set $S=\{C_i\}_{i=1...n}$, computes a score from v and finds the color $C_f \in S$ the furthest from the average. If this color is further than a defined distance d, then it is removed from S. This step is repeated until stability or until S contains only 2 elements. The returned score is the variance found in the last step. The choice of the constant d depends on the input cameras layout and on the scene complexity. This algorithm can be summarized as follows :

- **bool** *stable* = **false**
- $S = \{C_i\}_{i=1...n}$
- a = average(S)
- v = variance(S, a)
- *score* = scoreFunction(v, Card(S))
- **do**
  → find the farest color $C_f \in S$ from a
  → **if** distance(C_f, a) ≥ d **then**
  - $S = S - C_f$
  - a = average(S)
  - v = variance(S, a)
  - *score* = scoreFunction(v, Card(S))

  else *stable* = **true**

  while Card(S) ≥ 2 **and** *stable* = **false**

The *scoreFunction* weighs the variances according to *Card(S)* such that with equal variance, the set of colors with the maximum cardinal is favoured. A good score corresponds to a small variance.

Finally, we propose a third method to compute the colors' scores. This method also begins by a variance and an average computation in the color set

$S=\{C_i\}_{i=1...n}$. Then we find the color $C_f \in S$ that is the furthest from the average. A new variance and a new score are computed without this color. If this score is better than the previous one, C_f is removed from S. This step is repeated until a good score is found or until S contains only 2 elements. The score is set as the variance weighed by the cardinal of S. This algorithm can be summarized as follows :

- **bool** *stable* = **false**
- $S = \{C_i\}_{i=1...n}$
- a = average(S)
- v = variance(S, a)
- *score* = scoreFunction(v, Card(S))
- **do**
  → find the farest color $C_f \in S$ from a
  → $a^* = \text{average}(S - C_f)$
  → $v^* = \text{variance}(S - C_f, a^*)$
  → $score^* = \text{scoreFunction}(v^*, \text{Card}(S)-1)$
  → **if** $score^* \leq score$ **then**
  - $a = a^*$
  - $v = v^*$
  - $score = score^*$
  - $S = S - C_f$
  else *stable* = **true**
  while Card(S) ≥ 2 **and** *stable* = **false**

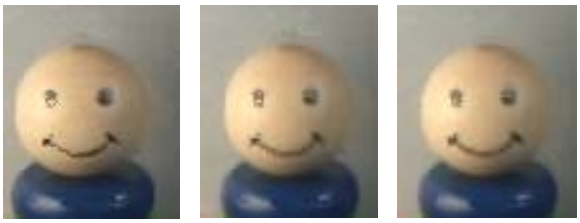
These three methods are easily implemented using fragment shaders. As shown in Figure 4, the two iterative methods provide better visual results, especially when the input camera are placed in a 1D arc configuration which increase the occlusions effects.

(a) (b) (c)

Figure 4: image (a) is computed using the variance and the average, (b) using the sigma clipping technique and (c) using the second iterative method.

Neighborhood with mipmapping

For more robustness during the scoring stage, we take into account the neighborhood color contribution of each pixel. Mipmapping provides access to the same image but with a lower level of details (lod) and hence provides the average color of the neighborhood of the current pixel. For each pixel



(a) (b) (c)

Figure 5: Images computed with different mipmap levels : (a) no additional mipmap level, (b) 1 mipmap level and (c) 2 mipmap levels.

score, we combine the score computed using different lods. Yang et al. [YWB02] propose a summation over a box-filtered lod pyramid but only one additional mipmap level works well with our method and more mipmap levels do not improve the visual results. This is illustrated in Figure 5.

Stereoscopic rendering

Virtual reality applications often requires stereoscopic display to increase immersion and most of these applications have to render the scene twice. But a lot of information such as diffuse lighting for example can be shared for both views. Concerning IBR techniques, depth-mapping is often view-dependant and hence the two new views must be computed separately. The plane-sweep algorithm computes local score associated to scene points. This information can be shared for several virtual cameras. We extend our method with a low cost algorithm providing the second view.

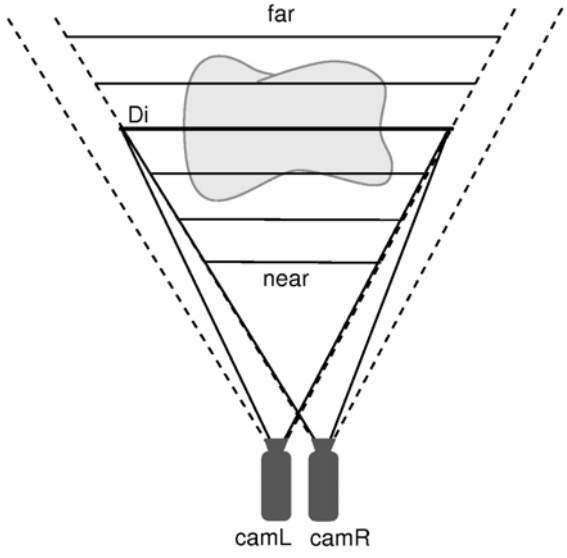


Figure 6 : Each plane D_i is common to the two views, but their projection differs

Stereoscopic rendering must satisfy several conditions concerning virtual camera parameters

[SC97]. In particular, both cameras must have their principal ray parallel to avoid vertical parallax in the stereoscopic image. Let cam_L and cam_R be a pair of virtual cameras satisfying this constraint and $D_{1...m}$ a set of planes parallel to these cameras' image plane. As shown in Figure 6, the score and the color computation of a plane D_i is common for both cam_L and cam_R. Only the projection of D_i on the two cameras will differ. The score computation is a central task in the plane-sweep algorithm, so sharing this stage among the two views provides a consequent gain in computation time. Thus, our plane-sweep method must be modified as follows :

- initialize I_L and I_R's score
- **for** each plane D_i from *far* to *near*
  - → project all the input images $I_1...I_n$ on D_i
  - → render D_i on two textures *texScore* and *texColor* : **for** each pixel of I_{tmp}
    - compute a score and a color according to the coherence of the colors from each camera's contribution
  - → copy *texScore* and *texColor* on D_i
  - → project D_i multi-textured on I_L and I_R
  - → **for** each pixel of I_L and I_R
    - **if** the score is better than the previous one **then** update the score and the color
- draw I_L and I_R

For each planes D_i, this method first computes scores and colors and stores them in two textures. In a second pass, these two textures are copied on D_i and projected on the two virtual cameras. The first pass requires off-screen rendering performed by Frame Buffer Objects (FBO) and Multiple Render Target (MRT). This step can also be achieved using p-buffers with a small frame rate penalty.

Thus, this method can easily be implemented such that all the image data stay in the graphic card and hence avoid expensive data transfers between the graphic card and the main memory.

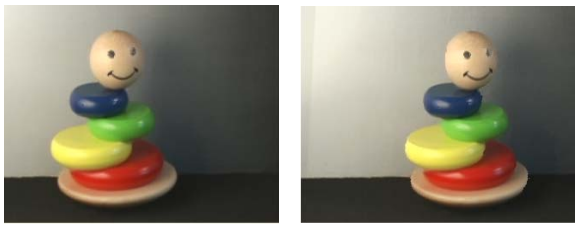


Figure 7: Real-time stereoscopic pair (cross vision)

Figure 7 shows a stereoscopic pair rendered in real-time. Note that the fusion of the two images decreases the imperfection impact of the images.

As illustrated in Table 1, stereoscopic rendering achieves a 15% frame rate decrease instead of the 50% expected by rendering twice the scene.

Implementation

Input cameras are calibrated using the *gold standard algorithm* [HZ04]. We implemented our method on OpenGL 2.0 and we use OpenGL Shading Language for the scoring stage.

For more accuracy, the texture coordinates are computed using projected textures directly from the camera projection matrices. We use multitexturing in order to get access to each texture during the scoring stage. Each score is computed with fragment shaders using mipmapping. They are stored in the `gl_FragDepth` and the colors in the `gl_FragColor`. Hence we let OpenGL select best scores with the z-test and update the color in the frame buffer.

To compute a depth-map rather than a new view, we just set the `gl_FragColor` to the `gl_FragCoord.z` value.

Most of the work is done by the graphic card and the CPU is free for others tasks.

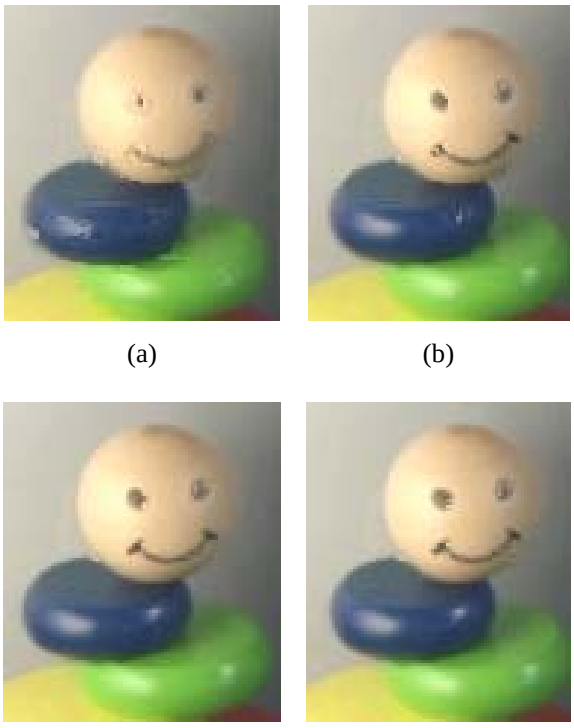
5. RESULTS

We tested our methods on an Athlon AMD 1GHz with a Nvidia GeForce 6800GT. We used four tri-CCD Sony DCR-PC1000E cameras for the input images acquisition. The white balance is essential in a plane-sweep algorithm. Indeed, we must homogenize the camera color range such that any point in the scene is seen with the same color from each camera. For our tests, we used the manual white balance provided by the tri-CCD cameras but for more accuracy, we planed to use a color calibration method as proposed by Magnor [M05].

Table 1 shows the framerate we obtain with 4 input cameras.

Number of plans *D*	Simple variance	Method 1 and 2		Stereo-scopic
10	140	85	91	110
30	43	28	30	38
50	30	17	18	25
100	15	9	9	13

Table 1. Frame rate in frame per second for a 320x240 image from 4 input cameras.



(a) (b)

(c) (d)

Figure 8: Number of planes used for each scene : (a) 5 planes, (b) 10 planes, (c) 30 planes and (d) 50 planes

The computation time depends on the number of planes we choose to discretize the scene. Our tests indicate that after 50 planes, the quality difference becomes neglible (Figure 8).

We are presently working on examples of on-line dynamic scenes.

6. CONCLUSION

This paper presents a plane-sweep method that allows real-time rendering of on-line dynamic scenes. Except for *near* and *far* planes, it does not require any prior knowledge of the scene. This method can be implemented on every consumer graphic harware that supports fragment shaders and therefore frees CPU for other tasks. Furthermore, our scoring method enhances robustness and implies fewer constraints on the position of the virtual camera, *i.e.* it does not need to lie between the input camera's area.

We propose to extend our research in optimisation of D_i planes repartition in order to reduce its amount without depreciating the visual result. We also intend to achieve a better stereo viewing result by producing pairs of virtual cameras with non symetric projection pyramid in order to save space on the edges of the stereo images [GPS94].

7. REFERENCES

[BBH03] Myron Z. Brown, Darius Burschka, and Gregory D. Hager. *Advances in Computational Stereo*, IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 993-1008, 2003.

[C96] Robert T. Collins, *A Space-Sweep Approach to True Multi-Image Matching*, in proc. Computer Vision and Pattern Recognition Conf., pages358-363, 1996.

[FWZ03] Andrew Fitzgibbon, Yonatan Wexler and Andrew Zisserman, *Image-based rendering using image-based priors*, 9th IEEE International Conference on Computer Vision (ICCV 2003), pages 1176-1183, 2003.

[GGSC96] J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen, *The lumigraph*, SIGGRAPH, pages 43-54, 1996.

[GKV04] Indra Geys, T. P. Koninckx and L. Van Gool, *Fast Interpolated Cameras by combining a GPU based Plane Sweep with a Max-Flow Regularisation Algorithm*, in proc. of second international symposium on 3D Data Processing, Visualization & Transmission - 3DPVT'04, pages 534-541, 2004.

[GPS94] V.S. Grinberg, G. Podnar and M. Siegel, *Geometry of Binocular Imaging*, Stereoscopic Displays and Virtual Reality Systems, Vol. 2177, pages 56-65, 1994.

[GV05] Indra Geys and L.Van Gool, *Extended view interpolation by parallel use of the GPU and the CPU*, in proc. of IS&T SPIE, 17th annual symposium on electronic imaging - videometrics VIII, vol. 5665, pages 96-107, 2005.

[HZ04] Richard Hartley and Andrew Zisserman, *Multiple View Geometry in Computer Vision*, second edition, Cambridge University Press, ISBN: 052154051, 2004.

[IHA02] M. Irani, T. Hassner and P. Anandan. *"What does the scene look like from a scene point?"*, Proc. ECCV, pages 883-897, 2002.

[KRN97] Takeo Kanade, Peter Rander and P. J. Narayanan, *Virtualized Reality: Constructing Virtual Worlds from Real Scenes*, IEEE MultiMedia, volume 4, pages 34-47, 1997.

[LH96] Marc Levoy and Pat Hanrahan, *Light Field Rendering*, SIGGRAPH, pages 31-42, 1996.

[M05] Marcus A. Magnor, *Video-Based Rendering*, Editor : A K Peters Ltd, ISBN : 1568812442, 2005.

[MB95] Leonard McMillan and Gary Bishop, *Plenoptic Modeling: An Image-Based Rendering System*, SIGGRAPH, pages 39-46, 1995.

[MBRGM00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler and Leonard McMillan, *Image-Based Visual Hulls*, in proc ACM SIGGRAPH, pages 369-374, 2000.

[PKV00] M. Pollefeys, R. Koch, M. Vergauwen and L. Van Gool, *Automated reconstruction of 3D scenes from sequences of images*, ISPRS Journal Of Photogrammetry And Remote Sensing (55)4, pages 251-267, 2000.

[SC97] StereoGraphics Corporation, *Developer's Handbook : background on creating images for CrystalEyes and SimulEyes*, StereoGraphics Corporation, 1997.

[SLS01] Hartmut Schirmacher, Ming Li and Hans-Peter Seidel, *On-the-fly Processing of Generalized Lumigraphs*, Proc. EUROGRAPHICS 2001, Eurographics Association, pages 165-173, 2001.

[SS02] Daniel Scharstein and Richard Szeliski,*A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*, IJCV, 47, pages 7-42, 2002.

[WJKR04] Woetzel, Jan, Koch and Reinhard, *Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware*, in proc. of 17th International Conference on Pattern Recognition (ICPR 2004), pages 741-744, 2004.

[YWB02] Ruigang Yang, Greg Welch and Gary Bishop, *Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware*, in proc. of Pacific Graphics, pages 225-234, 2002.

[ZBUWS04] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder and Richard Szeliski, *High-quality video view interpolation using a layered representation*, in proc. ACM SIGGRAPH, pages 600-608, august 2004.