A Parallel Multi-View Rendering Architecture

Wallace Lages, Carlúcio Cordeiro, Dorgival Guedes

Deep Computing Visualization Center
Universidade Federal de Minas Gerais
Belo Horizonte, Brasil
wlages@ufmg.br, carlucio@dcc.ufmg.br, dorgival@dcc.ufmg.br

Abstract

We present an architecture for rendering multiple views efficiently on a cluster of GPUs. The original scene is sampled by virtual cameras which are used later to reconstruct the desired views. We show that this image-based approach can be very scalable and support rendering at interactive rates.

1. Introduction

Multiple views are typically used to give sense of depth and perspective for stereo and autostereoscopic displays [3], [10], [11], [13],[25]. However, multiple viewpoints are also needed if we want to support several users visualizing a given dataset at the same time. Among these applications we may include: engineering projects, virtual reality environments, geographic models and massive computer games. Each user is connected to the rendering cluster by a thin client and is interested on a particular view of the stored dataset.

To be able to render hundreds or thousands of views at interactive rates, we need a solution which is both scalable and efficient. Clusters of GPUs have become an attractive platform for rendering tasks, since they have an excellent price-to-performance ratio and are easily available.

In this paper we present a parallel architecture designed to render multiple views interactively on a cluster of GPUs. The main challenges are to obtain an efficient parallelization, minimize the overhead due to communication and scale as both more users and more PCs are added to the system.

Our idea is to combine sampler nodes with image-based renderers so that computation can be parallelized and reused among the views. We analyze the theoretical speedup with respect to the number of views and provide experimental evidence that the proposed architecture is feasible for the task.

This paper is organized as follows: In section 2 we discuss relevant related work. In section 3 we describe the architecture and parallelization strategy, in section 4 we explain details of the light field renderer. In section 5 we provide experimental results that support our findings. In section 6 we discuss future work and conclude.

2. Background and Related Work

Following we give a brief overview on multiple viewpoint rendering and parallel rendering.

2.1. Multiple Viewpoint Rendering

Rendering multiple views using the standard graphics pipeline is a difficult problem. The current rasterization architecture requires an early decision of the viewpoint, which makes it difficult to explore coherence between the views. For this reason, Halle [10] suggested rendering the views into a spatio-perspective volume, created by rendering epipolar images. Hübner et al. [13] developed a multi-view splatting technique which renders the necessary views for each splat just one time. They use the GPU to project each splat to all views at once.

Other attempts in multi-view rendering involved new hardware. Hasselgren and Akenine-Möller [11] proposed an architecture capable of rendering each triangle to multiple views simultaneously, improving texture access coherence. Stewart et al. developed PixelView [25], a hardware prototype that employed a 4D frame-buffer to support view selection at the time of raster scan-out. Although these propositions allow us to explore new algorithms, they have limited use in the current available hardware.

The concept of using a sampled 4D buffer to generate new views is known as lumigraph or light field rendering. A light field [16] is a dense sampling of the 5-dimensional plenoptic function [1], which describes the radiance of every point in space (P_x, P_y, P_z) in all directions (θ, ϕ). If

we consider only the subset of rays leaving a bounded object, the function can be further reduced to 4 dimensions [9]

Besides being used by Stewart et al. [25], light fields have been used in the literature to generate new views from a camera array [31]. In a certain sense the buffer proposed by Halle [10] can also be considered a light field. However, in our work, rather than constructing and sampling the entire light field for each new view, we are only interested in using it as a way to share rays among views and avoid duplicated computational effort.

Even using ray sharing, the cost of rendering multiple views interactively is still larger than the power available from a typical graphics card. For this reason, we propose a scalable parallel architecture. Annen et al. [3] describes a distributed system for rendering multiple views for parallax displays. However, they do not explore any sort of coherence in the views, and focused only at scalability and load balancing. Yang et al. [31] developed a system that distributes some of the steps necessary to reconstruct a single view, achieving scalable bandwidth costs. Similar to ours, they do not use a ray buffer, so it can be applied to dynamic scenes.

2.2. Sampling Issues in light field rendering

The minimum number of samples required to reconstruct a signal is a classic problem in image processing and computer graphics. Sampling analysis in image based rendering is a difficult problem because it involves a complex relationship among three elements: the depth and texture information of the scene, the number of sample images, and the rendering resolution [6], [17],[18].

It is well known that depth information, besides improving rendering quality, also affects sampling rate. Since per-pixel depth information is free on synthetic images, it has been used on some recent work [22],[28].

It is also important to use a good sampling structure. For objects it is common to use a spherical parametrization [14],[28]. For environments this is not so clear. The simplest approach is to sample the scene using a regular grid [23]. Another possibility is to generate samples adaptively as in [15].

2.3. Parallel Rendering

Parallelism has been used many times for accelerating rendering. One can either divide the rendering steps among the processors (pipelining) or split the rendering data. Another possible option is to explore the time dimension.

Following the taxonomy proposed by Molnar et al. [21] we can classify the rendering architectures by the time when visibility is solved. *Sort-first* architectures divide the final image in disjoint tiles. Each node is responsible for rendering one or more tiles which are easily composed afterward. In *sort-last* architectures, each node renders a portion of the primitives into an image, recording the z value for each pixel. Later, all images are composited into a final image using the z values of each one. In the *sort-middle* solution, the primitives are first distributed to geometry processors to be transformed and projected. After projection they are redistributed again among rasterization processors. As the sort-middle approach requires reading back the projected geometry it is not well suited to GPUs with dedicated pipelines.

Sort first architectures usually have smaller communication requirements, but they suffer from a higher processing cost, caused by duplicated rendering among tiles and the transforming necessary for sorting. On the other hand, sort last renders each primitive exactly once, without transforming them. This makes it more scalable than the sort first approach. As a disadvantage, the composition step is more complex and the bandwidth requirements are greater. Several systems and parallelization strategies have been proposed along the years for both sort-last and sort-first approaches. For a summary please refer to Crockett [7].

More relevant to our work is the parallelization of image based renderers. Sloan and Hansen [24] present three techniques for reconstructing lumigraphs in parallel. However, they use target a distributed shared memory ccNUMA computer. A more recent work has been developed by Strasser et al. [26]. The MLIC system uses an image cache to decouple image rendering from visualization and distribute the rendering calculations among multiple computers. They decompose the space around the view-point into six pyramids. Images are placed at fixed positions with respect to the view-point and may be further refined with a kd-tree. Displaying the image database involves visiting all the polyhedra, and traversing the accompanying kd-trees in a top-down manner and back-to-front.

Many hybrid geometry/image systems have also been proposed for parallel rendering of massive models. The basic idea is to render objects far from the viewpoint using fast image-based techniques. Among them we may cite the work of Wilson and Manocha [30], Aliaga et al. [2] and Debevec et al. [8].

3. Parallel Multi-view Rendering

The time required to render a number of views V using the standard rendering pipeline is gV, where g is the time to render a single view:

$$T_{naive} = (gV)$$

To render multiple views in a scalable fashion, we need a parallel architecture. The most naive parallelization is to

divide the views among the available processors P, so that the time between two views is:

$$T_{||naive} = (gV)/P \tag{1}$$

We can assume that the load is evenly distributed since an image can be sliced between two or more processors. This architecture is depicted in Figure 1.

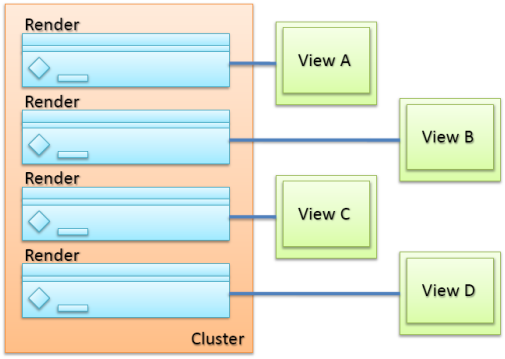

Figure 1. Intuitive Architecture - The views are distributed among the available GPUs. Each GPU render the entire pipeline.

However, to improve the performance while rendering multiple views, we would like to reuse computation that can be shared among the views. We propose using a light field renderer to render the views from scene samples instead of the original geometry. If S is the number of images necessary to sample the environment and c the time to compose a new image from the samples, we can split the gV into a sampling and compositing steps, so that:

$$T_{new} = (Sg + Vc)$$

This formulation is faster than T_{naive} above a certain number of views as long as $c < g$. The efficient parallelization of this new algorithm, requires a change in the architecture, since all compositing tasks must access the samples being generated. We split the processors into a 2 level superscalar pipeline. The first level is responsible for generating the samples, the second for compositing the views (Figure 2).

From the P processors available, K are used to sample the environment while $(P - K)$ are used to compose the final views. In the new system, the time between the ouput of two consecutive images is equal to the time taken by the slowest stage.

$$T_{||new} = Max\left(\frac{Sg}{K}, \frac{Vc}{P-K}\right) \tag{2}$$

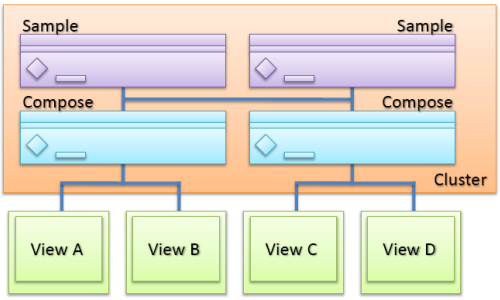

Figure 2. Proposed Architecture - The pipeline is divided into sampling and compositing stages. The GPUs are assigned to each stage as a function of the number of views.

We know that $T_{||new}$ is minimum when both stages takes the same time. However, if we impose the additional restriction that $K \in \mathbb{Z}$, a perfect balancing can not be achieved for all V. In this case the optimal K can be found by equating $T_{||new}$ for K and $K - 1$. The expression for the a discrete setting is:

$$K_{fd}(S, g, V, c, P) = \left\lfloor \frac{SgP + Vc}{Vc + Sg} \right\rfloor \tag{3}$$

Since the new rendering formulation (T_{new}) can be used to save computation on a serial processor, it will be used to derive the speedup S for both systems:

$$S_{naive} = \frac{T_{new}}{T_{||naive}} = (Sg + Vc) \cdot \frac{P}{gV} \tag{4}$$

$$S_{new} = \frac{T_{new}}{T_{||new}} = (Sg + Vc) \cdot Min\left(\frac{K}{Sg}, \frac{P-K}{Vc}\right) \tag{5}$$

Since we are assuming $c < g$, $S_{naive} < S_{new}$ above a certain number of views.

4. Light field Rendering

Our light field renderer builds upon the work on per-pixel depth information for the correction of rays during reconstruction [22], [28], [12], [29].

There are many ways of parameterizing the 4D light field. The best one depends on the application domain, but they usually possess good sampling characteristics and can ease the reconstruction task. Among the parameterizations proposed in the literature we may cite: two planes [16], cylindrical [20], spherical [28], [14], two spheres, plane-sphere [5] and non structured [22], [4], [27].

For this paper we chose to use a heightfield terrain. For this reason, we decided that a 2 plane parametrization (2PP) would be a good fit. The 2PP index each ray entering the scene by a pair of points (u, v) and (s, t) on the two planes. The (s, t) plane is known as view point plane and the (u, v) as image plane.

4.1. Acquiring depth images

Each block of the scene is enclosed in the volume defined by the planes $z = 1$, $z = 0$, $x = \pm 1$ and $y = \pm 1$. Four cameras are at the positions $(-1, -1, z)$ $(-1, 1, z)$, $(1, -1, z)$ and $(1, 1, z)$ At each frame, the 4 cameras render the scene (Figure 3) and store the depth in the alpha channel as:

$$d = \frac{|P_g - P_1|}{|P_1 - P_0|}$$

where P_1 and P_0 denotes the intersections of the viewing ray with the plane $z = 1$ and $z = 0$ respectively. P_g is the point where the viewing ray intersects the geometry (This is demonstrated in Figure 4, for the ray exiting camera Ca).

4.2. Rendering new views

The rendering algorithm was implemented in a GLSL fragment program. To render a new view, we need the position of the viewing camera C_{vp}, the position for each of the k sample cameras C_{kp} and the image plus depth captured by each one C_{ki}. We also need a projection matrix $Proj$ to specify the camera model. In this work we consider that all the cameras are always looking to the center of the tile, so orientation information is not needed.

To be able to use the fragment shaders, we need a render surface. So, for each tile described on the previous section, we draw a quad. The quad is parallel to the plane $z = 0$ and positioned at the origin. This will guarantee a render surface as long as the desired camera has a Z coordinate greater than the near clipping distance.

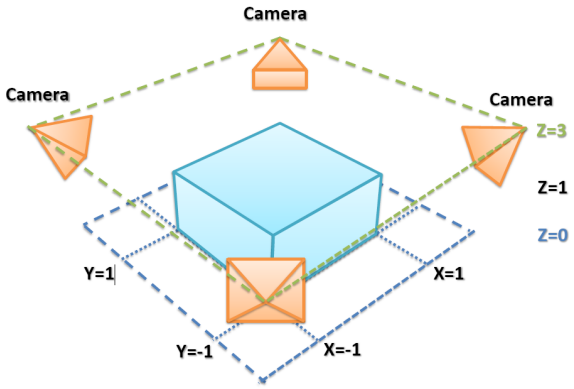

Figure 3. Four cameras are used to sample each tile. The sample volume is delimited by the planes $z = 0$, $z = 1$, $x = \pm 1$, $y = \pm 1$

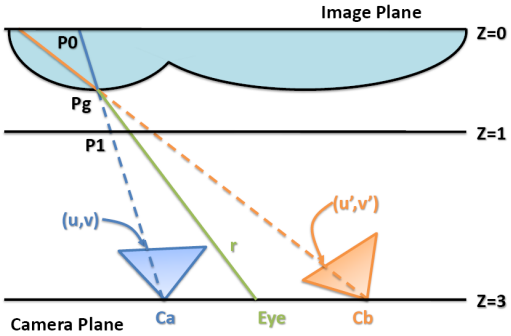

Figure 4. Given ray r, we must find coordinates (u,v) and (u',v') which correspond to intersection point P_g

To render a new view, we must decide for each pixel, which samples to use. If the depth is known, we can use image warping to project pixels from one view to another[19]. This can be done performing a search along the view ray, reprojecting the point into each camera and comparing the distance with the value stored in the image [28].

First, we need to compute the viewing vector for each pixel. This can be computed passing the vertex position as an interpolated variable from the vertex shader:

$$v = normalize(VertexPos - D_v)$$

Next we need to compute the full projection matrix for each camera C_{kp} This can be achieved composing the pro-

jection matrix $Proj$ with each camera position and orientation.

$$Proj_k = Proj * rot_k * offset_k$$

If more than one camera is looking at the same point, we choose one of the samples.

We chose to apply inverse warping so that we could use GPU interpolation and avoid holes in the image. In order to find the correspondence between the cameras we need the depth of each pixel in the new image. There are many ways to obtain that, but they all involve some kind of search along the view ray v, for the intersection with the surface implicitly defined by the other cameras. For the sake of simplicity, we decided to use a linear search.

The search starts at the point where v intersects the plane $z = 1$. At each iteration, the current point can be expressed by:

$$P = plane.s1 + v * \delta$$

where

$$\delta = \frac{|plane.s - plane.s1|}{steps}$$

and $plane.s1$ and $plane.s$ correspond to the intersections between v and $z = 0$ and $z = 1$ respectively.

To find the intersection with the surface, we employ an approach similar to the one used by Todt [28]: at each step we project the current point onto each camera k. This will yield a (x, y) pair that can be used to retrieve the surface position C_{ks} stored for each camera along the ray C_{kp}

$$C_{ks} = plane.s * C_{ki}.depth + (1 - C_{ki}.depth) * plane.s1$$

However, instead of comparing the position P with the estimate C_{ks}, we compare the distance from C_{kp} to the current point P and the distance from C_{kp} to the surface C_{ks}.

if $(|C_{ks} - C_{kp}| - |P - C_{kp}|) < 0$ the point P has entered the surface and we can refine the position with a binary search.

We don't interrupt the search until P is considered inside the surface for all the camera estimates C_{ks}. At that point we choose the sample with the more conservative estimate (the one closer to the surface).

5. Experimental Results

In this section we report the data collected to investigate the speedup, efficiency, communication costs and load balancing for a real implementation. We have implemented the system described in this paper on a Linux cluster, composed of 11 dual-Pentiums 3.40GHz 64bits, connected by a Gigabit ethernet. Each PC has 3GB of memory and a Nvidia GeForce 7900 GTX/PCI/SSE2. We used OpenGl and GLSL for graphics and LAM/MPI: 7.0.6 for interprocess communication.

As input database we used a scene composed of one highly tessellated tree over a heightfield terrain. The scene has approximately 500 thousands triangles and was lit by a simple phong shader. During the simulation we logged:

1. The time required for rendering a view from the original data
2. The time required for rendering a view from the obtained samples
3. The time spent transferring between GPU and CPU
4. The time spent transferring data over the network
5. The total running time

For each camera we rendered 5 frames, moving the viewpoint in them. The samples were taken with a 45 degree projection and a resolution of 400x300 pixels. Output views were rendered at 200x150 resolution. To improve the throughput, network transmission was intercalated with rendering. The sample images were transmitted uncompressed using point to point communication.

In the tests involving a variable number of GPUs, we used K_fd (eq. 3) to ensure the best distribution among the stages. An output of the system is shown in Figure 9.

Performance

We measured the total rendering time (see Figure 5) and the time spent in each sub-task (see Figure 6) for an increasing number of views. Our system rendered 290 views in 2.61 seconds, spending approximately 9 ms per frame with 5 GPUs. The GPU/CPU communication costs were negligible. The time spent in MPI receive routines increased when more nodes were allocated to sampling. This was expected, since we used a point to point protocol. A collective function would help to keep this from growing too fast but this is not a problem since the sampling costs are independent from the number of views.

Speedup, efficiency and scalability

To compute the speedup we compared the total running time against the time a naive parallelization would take in a perfect implementation. As discussed in Section 3, this cost is equal to (gV). Figure 7 shows the results for 1000 views. The measured efficiency for 1000 views in Table 1 and a scalability comparing our approach against the naive one is in Figure 8

Rendering Quality We have compared image quality by rendering the original geometry for some views. Ideally these images would be equal to the ones generated by the light field renderer. However, the fidelity depends on the sampling rate used. In all the tests we used 4 cameras, which happens to be enough for our scene. The image quality also

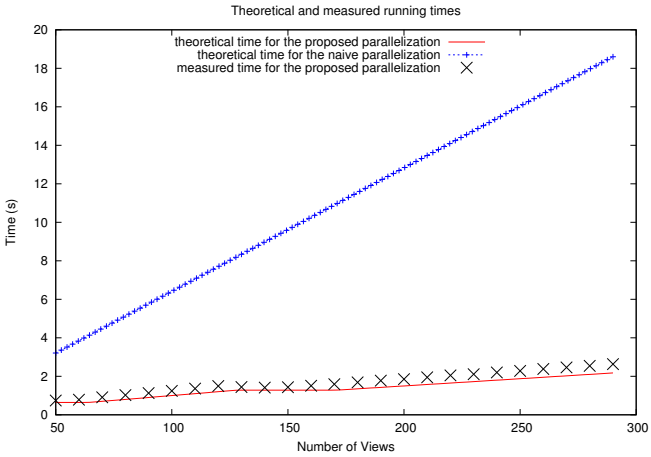

Figure 5. Time spent by the naive and proposed parallelizations using 5 GPUs

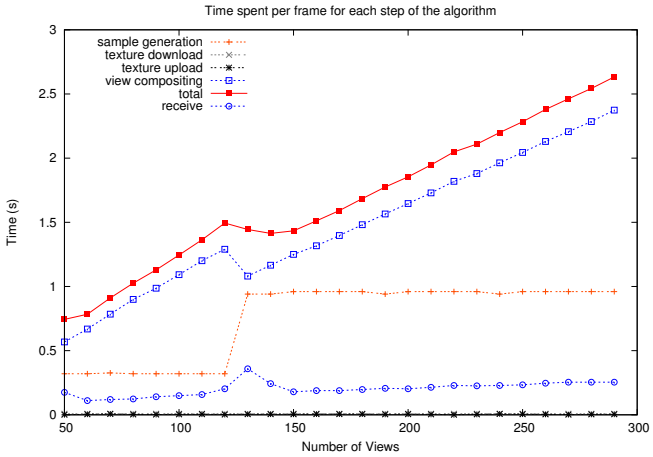

Figure 6. Time spent in the main steps of the algorithm

Total GPUs	Efficiency (Proposed)	Efficiency (Naive)
5	0.715	0.103
6	0.757	0.105
7	0.792	0.106
8	0.812	0.108
9	0.818	0.108
10	0.826	0.110
11	0.829	0.110

Table 1. Efficiency of Proposed and Naive Parallelizations for 1000 views. Number of GPUs allocated to sampling: 1

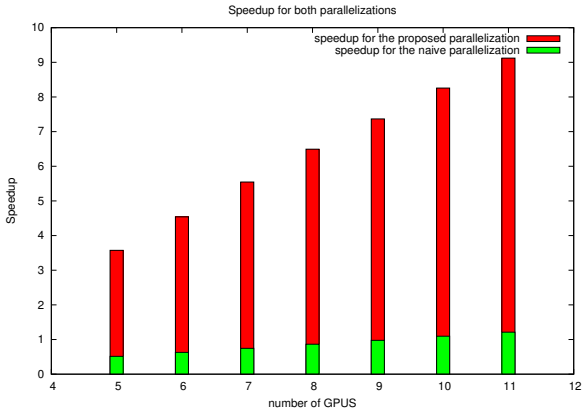

Figure 7. Speedup of the naive and proposed parallelizations for a load of 1000 views.

depends on the number of steps used to refine the geometry intersection. For the quality tests we used the Stanford Bunny model [1] and 100 steps.

To compare the reference and the rendered images, we used the Perceptual Image Diff Utility [2]. This utility makes use of a computational model of the human visual system to compare two images. A comparison between an image produced by our method and one rendered from original geometry is shown in Figure 11. Visible differences are highlighted in red.

1 http://graphics.stanford.edu/data/3Dscanrep/

2 http://pdiff.sourceforge.net/

6. Future work and Conclusion

In this paper we presented a parallel architecture for rendering multiple views in a cluster of GPUs. We have shown a theoretical analysis of speedup and scalability of the proposed system, showing how it can be superior to a naive parallelization. In the results, we have supported this analysis by running experiments on a rendering cluster.

The main insight of this work was the use of an image based renderer to avoid duplicated work among the views. At the same time, an efficient parallelization was found, what enables the system to scale very well with the num-

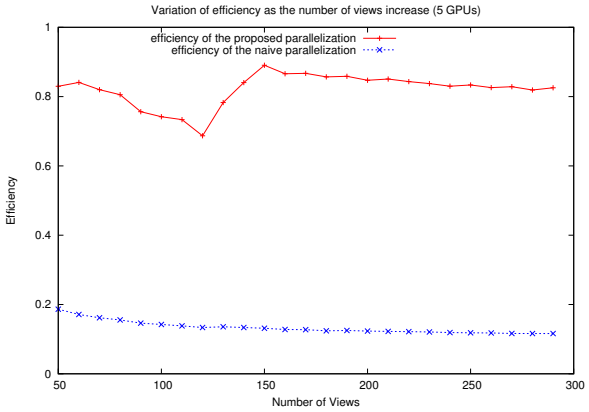

Figure 8. Scalability of the naive and proposed parallelizations

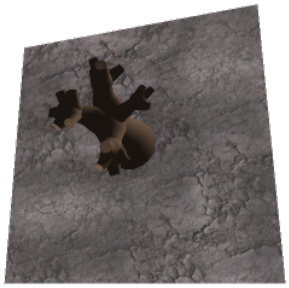

Figure 9. Render of the scene

ber of views.

The main tradeoff of the system is the sampling process, which may be complex for some scenes. In a future work, we would like to experiment with adaptive camera positioning, so that we can improve the sampling of more complex scenes.

Another area of work is the overall optimization of the system, using faster texture transfer primitives (pixel/frame buffer objects), dynamical load balancing and faster image composition, using better search strategies.

Acknowledgments The authors would like to thank João Luiz Campos and the Tecgraf Group at PUC-Rio for their support with the experiments. We also would like thank CAPES-Brazil for their financial support.

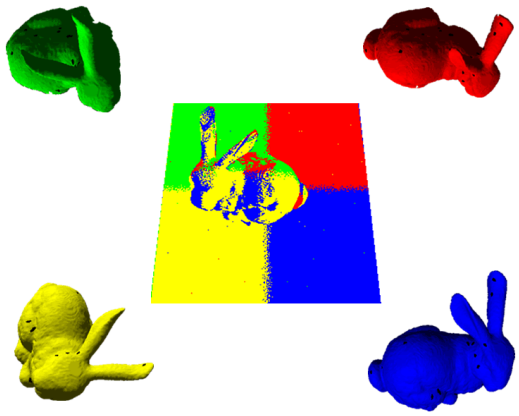

Figure 10. Input samples and contribution on the final image

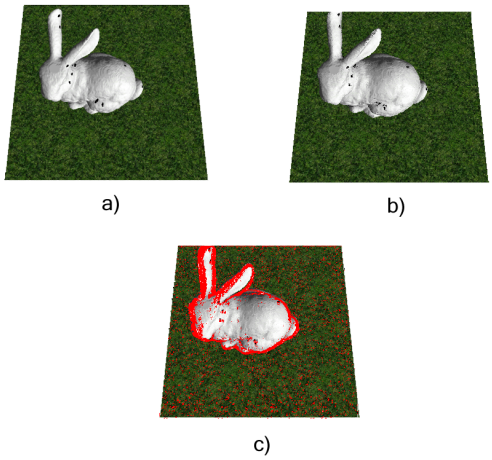

Figure 11. Output quality. a) render from the geometry; b) output of the light field, c) perceptual difference

References

[1] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. *Computational Models of Visual Processing*, pages 3–20, 1991.

[2] D. Aliaga, J. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. Mmr: an interactive massive model rendering system using geometric and image-based acceleration. In *I3D '99: Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 199–206, New York, NY, USA, 1999. ACM.

[3] T. Annen, W. Matusik, H. Pfister, and H.-P. Z. M. Seidel. Distributed rendering for multiview parallax displays. Technical report, Mitsubishi Electric Research Laboratories, Jan. 2006. "Distributed Rendering for Multiview Parallax Displays", SPIE Conference Stereoscopic Displays and Virtual Reality Systems XIII, Vol. 6055, pp. 231-240, January 2006, SPIE Proceedings.

[4] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen. Unstructured lumigraph rendering. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, New York, NY, USA, 2001. ACM.

[5] E. Camahort, A. Lerios, and D. Fussell. Uniformly sampled light fields. Technical report, Austin, TX, USA, 1998.

[6] J.-X. Chai, S.-C. Chan, H.-Y. Shum, and X. Tong. Plenoptic sampling. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 307–318, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[7] T. W. Crockett. Parallel rendering. Icase report 95-31; nasa cr-195080, Apr. 01 1995.

[8] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20, New York, NY, USA, 1996. ACM.

[9] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, New York, NY, USA, 1996. ACM.

[10] M. Halle. Multiple viewpoint rendering. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 243–254, New York, NY, USA, 1998. ACM.

[11] J. Hasselgren and T. Akenine-Möller. An efficient multi-view rasterization architecture. In T. Akenine-Möller and W. Heidrich, editors, *Eurographics Workshop/ Symposium on Rendering*, pages 61–72, Nicosia, Cyprus, 2006. Eurographics Association.

[12] W. Heidrich, H. Schirmacher, H. Kück, and H.-P. Seidel. A warping-based refinement of lumigraphs. In N. Thalmann and V. Skala, editors, *Proc. WSCG '99*, 1999.

[13] T. Hübner, Y. Zhang, and R. Pajarola. Multi-view point splatting. In Y. T. Lee, S. M. H. Shamsuddin, D. Gutierrez, and N. M. Suaib, editors, *GRAPHITE*, pages 285–294. ACM, 2006.

[14] I. Ihm, S. Park, and R. K. Lee. Rendering of spherical light fields. In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, page 59, Washington, DC, USA, 1997. IEEE Computer Society.

[15] S. Jeschke, M. Wimmer, H. Schumann, and W. Purgathofer. Automatic impostor placement for guaranteed frame rates and low memory requirements. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 103–110, New York, NY, USA, 2005. ACM.

[16] M. Levoy and P. Hanrahan. Light field rendering. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, New York, NY, USA, 1996. ACM.

[17] Z. Lin and H. Shum. On the number of samples needed in light field rendering with constant-depth assumption. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-00)*, pages 588–597, Los Alamitos, June 13–15 2000. IEEE.

[18] Z. Lin and H.-Y. Shum. A geometric analysis of light field rendering. *Int. J. Comput. Vision*, 58(2):121–138, 2004.

[19] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[20] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995.

[21] S. Molnar, M. Cox, D. Ellsworth, and H. Fuchs. A sorting classification of parallel rendering. Technical Report TR94-023, 8, 1994.

[22] H. Schirmacher, C. Vogelgsang, H.-P. Seidel, and G. Greiner. Efficient free form light field rendering, 2001.

[23] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 75–82, New York, NY, USA, 1996. ACM.

[24] P.-P. Sloan and C. Hansen. Parallel lumigraph reconstruction. In *PVGS '99: Proceedings of the 1999 IEEE symposium on Parallel visualization and graphics*, pages 7–14, Washington, DC, USA, 1999. IEEE Computer Society.

[25] J. Stewart, E. Bennett, and L. McMillan. Pixelview: A view-independent graphics rendering architecture. In T. Akenine-Möller and M. McCool, editors, *in: proc of Graphics Hardware*, pages 75–84, 2004.

[26] J. Strasser, V. Pascucci, and K.-L. Ma. Multi-layered image caching for distributed rendering of large multiresolution datasets. In B. Raffin, A. Heirich, and L. P. Santos, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, pages 171–177, Braga, Portugal, 2006. Eurographics Association.

[27] K. Takahashi and T. Naemura. Unstructured light field rendering using on-the-fly focus measurement. In *ICME*, pages 205–208. IEEE, 2005.

[28] S. Todt, C. Rezk-Salama, and A. Kolb. Fast (spherical) light field rendering with per-pixel depth. Technical report, University of Siegen, 2007.

[29] C. Vogelgsang and G. Greiner. Adaptive lumigraph rendering with depth maps. Technical Report 3, IMMD 9, Universitaet Erlangen-Nuernberg, 2000.

[30] A. Wilson and D. Manocha. Simplifying complex environments using incremental textured depth meshes. *ACM Trans. Graph.*, 22(3):678–688, 2003.

[31] J. C. Yang, M. Everett, C. Buehler, and McMillan. A real-time distributed light field camera. In *In: proc. of the 13th Eurographics workshop on Rendering, Italy*, pages 77–86, 2002.