The *Multi-LDI*: an Image Based Rendering Approach for Interaction, Navigation, and Visualization in Complex Virtual Environments

Stanislav L. Stoev and Ingmar Peter and Wolfgang Strasser

WSI/GRIS, University of Tübingen,
Auf der Morgenstelle 10, C9,
72076 Tübingen, Germany
{sstoev,peter,strasser}@gris.uni-tuebingen.de

ABSTRACT

In this paper, we present a new data structure for image-based rendering: the *multi LDI*. The multi LDI consists of a number of Layered Depth Images (LDI) covering a hemisphere of possible viewing angles. It allows compact image-based storage and fast rendering of large and complex scenes, while supporting rendering from large range of viewing directions. Since the internal occlusion in each LDI is very small, and only one of the LDIs in the multi LDI is rendered at a time, the rendering cost is significantly reduced compared to geometry-based rendering and even compared to other image-based rendering methods. Moreover, the single LDIs in the multi LDI can be generated on demand using a number of depth images rendered with an offscreen renderer. We also discuss a comparison of the geometry-based rendering and our image-based method and present some measured rendering times.
Furthermore, we describe the utilization of this technique in a complex Virtual Environment (VE) for realizing navigation, visualization, and interaction aids. In particular, we present a multiple viewport technique providing two important features: (1) A sort of history during the modeling process, whereas a live 3D copy of the scene is displayed in a window in front of the user. (2) Different *live* views of the scene, seen from *arbitrary* viewpoints, in order to display details occluded in the normal view.

Keywords: Image Based Rendering, Layered Depth Images, Virtual Environments, Interaction Techniques.

1 Introduction

The main objective of a virtual environment (VE) is to give the best possible impression of reality, while displaying virtual worlds. These might be distant worlds, micro/macro worlds, as well as artificial worlds. Keeping in mind the fact that many users are quite familiar with the desktop paradigm, the developers of virtual environments attempted to adapt some of the well-known and well working desktop techniques for interaction to VR. This is done in all of the available VR-concepts nowadays: head mounted displays (HMD), CAVE-like setups, and table-top projections.

A typical example for such an interaction is the multiple viewpoint technique. Viewing a scene simultaneously from multiple different perspectives is often used in CAD-systems. This in turn allows a better understanding of and orientation in the explored environment. A counterpart of such a tool in a 3D application is introduced in [Schma99] and is called *snapshot tool*.

Additionally to the adaptation of desktop techniques to virtual environments, synthetic environments allow the implementation of techniques not having a counterpart neither in desktop computers, nor in the physical world. For this, real-life human abilities have been intensively studied in the past decades. The results have been applied to improve existing and develop new human-computer-interaction techniques in VE. For instance, in a VE one can use virtual maps (or WIM which stand for world-in-miniature as introduced in [Stoak95]). Despite the common use of such techniques, however, virtual tools have also limitations. One of the most important issues is the rendering time. Even little decrease of the frame rate can significantly affect the user's perception and interaction.

In this paper, we consider the two techniques introduced above, the snapshot tool (Figure 1(a)) and the WIM-tool (Figure 1(b)), and propose an image-based rendering method for their realization in virtual en-

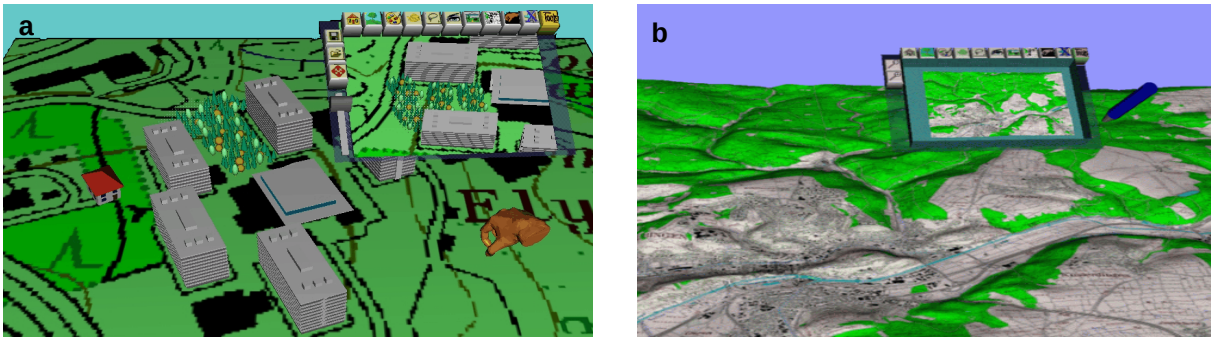


Figure 1: The world in miniature and the viewpoint freezing (*snapshot*-tool) tools.

vironments: the *multi LDI*. The common, straightforward implementation uses a reference or a copy of the virtual world. Unfortunately, the multiple rendering of the data can significantly deteriorate the frame rate and even negatively influence the interaction. The proposed multi LDI approach enables us to render fast not a copy of the scene, but a view-dependent, scene-complexity independent 3D image of it.

2 Related Work

The previous work discussed in this section is divided in two main related areas.

First, we focus on interaction in virtual environments in general and describe some background on navigation in virtual worlds. Afterwards, in Section 2.2 we discuss related work on image-based rendering techniques and their application in VR.

2.1 Navigation and Interaction

The navigation in virtual environments is a problem with increasing importance, since the size of virtual worlds is rapidly growing. Fortunately, people are confronted with the counterpart of this problem in every day life. This facilitates the exploration of this problem. In their work, Darken and Sibert [Darke93] presented a toolkit for navigation applying principles from real world navigation aids (e.g. maps). They also compare the strengths and weaknesses of such aids. Stoakley et al. [Stoak95] extended this work to three-dimensional maps. He defines navigation as a term covering two related tasks: movement through a virtual space and determining the orientation relative to the surrounding environment. Considering these two issues, Stoakley introduced the *World-in-Miniature* or WIM-technique. A WIM is a scaled down 3D copy of the virtual world displayed on an hand-held panel. Originally, the WIM was applied for interaction in virtual worlds, i.e. manipulating objects in space. Pausch et al. [Pausc95] extended this approach to provide a navigation tool. Due to its easy and intuitive application, WIM-like tools are widely used in virtual environments nowadays. Therefore, we introduced the two-handed WIM technique to our system, which is based on the pip and pen interaction metaphor presented in [Szala97, Schma99].

A discussion and evaluation of navigation tools can be found in [Ware90]. Bier [Bier86] describes in his work a set of techniques for scene composition. Mine [Mine95] discusses the fundamental forms of interaction in virtual environments, including movement, manipulation, and menu interaction.

2.2 Image-Based Rendering

In the past years, image-based rendering (IBR) techniques are becoming more and more popular. Their success is based on the fact that the rendering complexity of an image-based scene representation is almost independent on the geometrical complexity of the scene. Instead, it is heavily dependent on the resolution of the images used in the IBR representation. The increasing computer power nowadays, makes it possible to view large scenes with a reasonable resolution and interactive frame rates using image-based approaches. This makes IBR techniques very well suited for our work.

Various attempts to integrate IBR in virtual environments are reported the literature. For architectural walkthroughs [Aliag97] or city scenes [Wimme99] it is often sufficient to use pictorial information to replace distant parts of the scene. Instead of rendering the complete interior of a distant room seen through a door, the authors of [Aliag97] use a *portal texture*. It shows a picture of the room and is placed into the door opening and only replaced by real geometry when the user approaches the respective door. In [Raffe97, Raffe98] the portal textures,

which are “flat” images, are replaced by *depth images*. Depth images store to each pixel color additionally the disparity value of each pixel. This allows carrying out a correct perspective warp of all pixels with respect to the viewer’s position. In contrast to portal textures, the perspective distortion of the image is handled correctly. Problems with depth images occur, if parts of the scene are visible from the viewer’s position, which are not stored in the depth image. Therefore, Aliaga and Lastra [Aliag99] employed *Layered Depth Images (LDI)* [Shade98] to replace parts of a complex scene, which are distant from the viewer. Unfortunately, the time and memory requirements of their approach are very extensive, because all LDIs (form 180 to more than 5000 LDIs), for all possible view directions have to be calculated. Since this was reported to take from one to 28 hours, the LDIs are computed in a preprocessing step.

In general, an LDI [Shade98] is an image, in which every pixel represents a ray (Figure 2). Each ray stores an arbitrary number of depth pixels. Thus, occluded parts of the scene are stored in the LDI as well. In this way, an LDI permits to calculate various views of a scene without showing any gaps caused by incomplete object information. The rendering time and the memory requirements of an LDI depends linearly on the total number of all pixels stored in all rays of the LDI structure. In contrast to *layered impostors* [Decor99, Schau98], LDIs do not use a fixed depth resolution. In this way, LDIs provide high image quality, while requiring moderate amount of memory. Furthermore, the acquisition of LDIs is very simple. Using a set of depth images the spatial position for each source pixel can be calculated and the pixel inserted into the LDI structure. Pixels representing the same surface point are removed by simple comparison of the projected depth values, thus keeping only relevant information stored in the LDI.

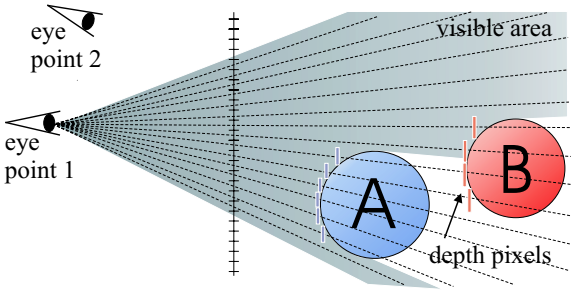


Figure 2: LDI storing two objects. The lower part of object B is invisible for eye point 1, but visible for eye point 2.

LDIs are widely used in various application areas, where complex geometry rendering is replaced by image-based rendering. For instance, in an *Image-Based Object* (IBO) [Olive99], six LDIs form a single object, which can be viewed from all directions. At most three of the LDIs, positioned on each side of a cube, are rendered to generate a correct view of the stored object.

3 The LDI-based Interaction Props

In this work, we present an extension of the transparent interaction props introduced by Schmalstieg et al. [Schma99]. Schmalstieg reports on the implementation of a snapshot mechanism for providing multiple views during the creation and exploration of virtual worlds. Once the scene seen through the transparent pip is fixed, a window with this content can be arbitrarily positioned in the space (a new viewport is defined). This multiple view tool, also called *snapshot-tool*, can be adjusted to always display a live-scene containing even objects added after the scene was “frozen”. *Freezing* means in this context that the scene, the user sees through the window, is fixed relatively to this window. In the next step, the window itself can be fixed in the space in front of the user (Figure 1(b) and 5(d)-(f)). Note, that *freezing* does not mean that the scene behind the window is static or a texture of it.

On the other hand, such a tool can also be used for freezing an unchangeable copy of the scene, reflecting particular stages of development. Besides its strengths, however, the main problem with the realization of this concept is that every time the world is caught on the pip, a reference or a copy of the latter is added to the rendered data. Thus, the original scene has to pass through the rendering pipeline an additional time, for each provided viewing window. Even though, these additional viewports are much smaller than the original viewport, the complete scene is rendered once for each additional viewport. Since this is unacceptable for large, detail-rich virtual environments, we employed an image-based technique for accomplishing this task (as will be shown in Section 5). The proposed technique allows for displaying the snapshot of the scene independent of its complexity, while supporting all features of the original snapshot tool.

Another contribution of this paper is the extension of the WIM-metaphor described above [Stoak95, Pausc95]. The problem is similar to the one introduced by the snapshot tool, namely the doubling of the rendered data. However, the WIM representing image-based structure has to be computed with significantly

higher resolution, in order to provide sufficiently detailed information when parts of the map are zoomed. Fortunately, the map tool does not have to be generated on the fly during the interaction, thus we compute it in a preprocessing step.

The props described above introduce several requirements, which have to be considered when choosing an appropriate image-based rendering technique:

- Data acquisition on-the-fly;
- Fast rendering of multiple views of the scene, covering a wide angle of viewing directions;
- Addition and removal of geometry objects to/from the scene.

In order to meet these requirements, we developed a new image-based data structure the *multi LDI*. A multi LDI consist of several small LDIs instead of providing one large LDI for covering the entire range of view directions (in our case a hemisphere). Since the memory requirements and rendering time depends linearly on the number of depth pixels stored in an LDI in total, the higher the amount of internal occlusion in an LDI, the less efficient will be the rendering. Apparently, the more the viewing position is altered the more visibility changes in the scene will occur. In the context of LDIs this means: The greater the range of viewing angles for which an LDIs allows the rendering of a correct image, the more depth pixels have to be stored in it and the less efficient will be the LDI rendering.

Since we have to cover the whole hemisphere of possible viewing angles above the snapshot tool respectively the WIM, storing all the scene information in a single large LDI will be quite inefficient. Therefore, the proposed multi LDI subdivides this hemisphere into patches and attaches one single LDI to each of them as shown in Figure 3. Each time a prop is rendered, the viewing angle relatively to the display window is used to determine the appropriate LDI. Since each of the LDIs covers only a small range of view directions, it can be rendered very fast.

In contrast to an IBO [Olive99], which requires the simultaneous rendering of up to three LDIs, with the multi LDI only one LDI is rendered at a time. Furthermore, each LDI of an IBO covers a wider range of viewing directions. Thus, the rendering time for each IBO's LDI is greater than the rendering time for one LDI of the multi LDI. Finally, the LDIs for an IBO have to be precomputed and cannot be generated on-the-fly, as this is done with the multi LDI as will be shown next.

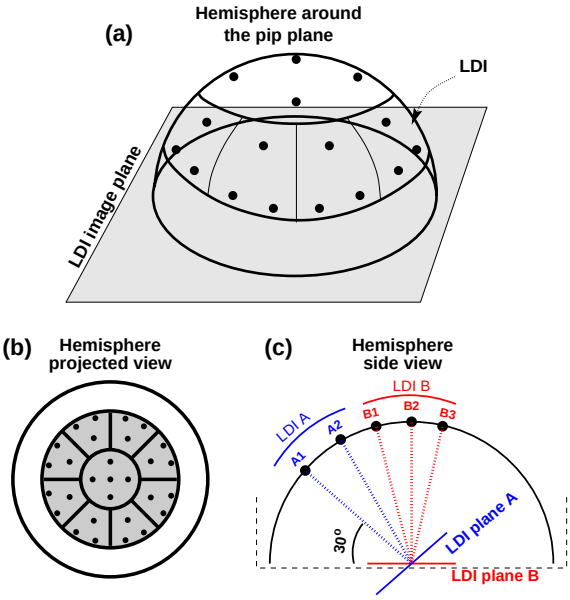


Figure 3: The dots indicate the camera positions used to generate the pictures for the single LDIs. The set of all LDIs defines the multi LDI. For the generation of the LDI A in (c), the depth images of the cameras $A1$, $A2$, and $B1$ are applied.

4 Hardware Setup

Our hardware setup consists of an electromagnetic 6 DOF tracker (*Ascension, Flock of Birds*) and a stereo table top display (*Barco, Baron*) called the *Virtual Table*. The tracker is used for determining the position and the orientation of three receivers. The first is used to track the position and orientation of the viewer's head. The virtual camera is attached to this receiver. The second is attached to a physical pen, which the user holds in his/her dominant hand. The virtual counterpart of the pen is used to manipulate the 3D virtual buttons, sliders, and other interaction elements projected on the pip. The pip is tracked with the third receiver and is a transparent panel, on which the interaction elements are back-projected [Schma99]. Furthermore, the pip is also used as an image plane for an additional viewport, through which the scene is projected. Additionally, the pip also allows to superimpose information through it, like this is done with the magic lens [Bier93].

5 Realization

In this section, we introduce the software environment used for our implementations. Afterwards, the image-based techniques for the snapshot tool and the WIM navigation tool are presented in detail.

5.1 Software System

Our implementation is based on the *Studierstube* framework [Schma96]. This is an object-oriented library extending the standard Open Inventor functionality. It allows for transparent processing of tracker events and their propagation through the scene graph.

5.2 Utilization

Initially, when the virtual map tool (WIM) is activated, the complete virtual world is mapped on the personal interaction panel of the user. Afterwards, the user can select a region of interest directly on the scaled down world, defining the scaling of the view volume [Stoev01]. A "beaming there" does not immediately follow this action. Instead, we offer a preview of the selected area through a seam mapped on the pip. First when the user finishes the current adjustment of the view area, he/she can activate the "beaming". Hence, a precise traveling through the virtual world is possible.

The application of the snapshot tool is easy as well. When this feature is activated, a reference (rather than a copy) of the current world is shown through the pip (a *second viewport* is defined as shown in Figure 5). Thus, the user does not see any difference between the scenes displayed through the two viewports. Now, the pen held in the dominant hand is applied as a virtual handle for adjusting the orientation, position, and scale of the world, shown through the second viewport [Stoev01]. When this adjustment is completed, one can freeze the world currently seen through the pip relative to the pip. This means that the pip (respectively the viewport on the pip) *and* the scene seen through the pip can be positioned in the space in front of the user. Since the creation of an LDI can be an expensive task, we left the initial SEAM-concept unchanged, thus geometry-based, until the user freezes the window in space. At this point, the generation of the multi LDI is triggered and it is displayed on the pip. In this way, the rendered geometry is not more than doubled. Each time a geometry containing SEAM-window is frozen, the LDIs for it are generated and no geometry is rendered. Thus, there can be no more than one geometry-rendered copy of the whole scene at a time.

5.3 LDI-acquisition

When the user freezes the new view window, we start with the LDI-generation. First, an imaginary hemisphere over the front side of the pip is aligned with the center of the pip (Figure 3 (a)). The radius of the hemisphere is set to be the current distance between the user's head position and the center of the pip. We divide the hemisphere into eight equally sized patches and a central patch, defining the multi LDI, as shown in Figure 3 (b). The patches cover 2/3 of the hemisphere's area corresponding to an angle of 120°. For each patch, which defines an interval of viewing directions, an LDI is created. The image plane (LDI-plane) of each patch is parallel to the plane defined by the three camera positions of the patch and contains the midpoint of the pip (Figure 3(c)).

The outer shell of the hemisphere is not covered with patches, since the view-angle in this area is very narrow and rarely used. In general the user views the window from the front, moving the virtual camera within the space covered by the LDI-patches. We ascertained that this is not limiting or distorting the utilization of the snapshot tool. However, in order to not confuse the user, we display a kind of an opened box (Figure 3(c)), which occludes the area not covered with LDI-patches before gaps can occur.

In the next step of the LDI-acquisition, we position a number of virtual cameras in each patch as depicted in Figure 3 (b). Each camera uses an offscreen renderer to generate a depth image of the scene (Figure 4). Knowing the position and

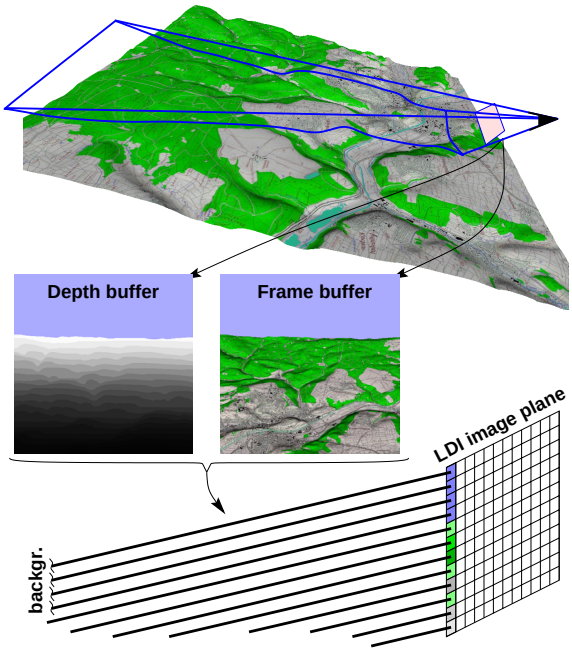


Figure 4: In order to create an LDI, we use the frame and depth buffer data of each shot picture and sort the pixels in the appropriate LDI(s).

orientation of the camera we can compute the position of each pixel in the LDI.

For the generation of a single LDI, we merge the depth images belonging to this patch (dots in each patch in Figure 3), as proposed by Shade et al. [Shade98]. In addition, the images of the cameras adjacent to the current patch are also taken into account. In this way, we can guarantee that no distorting "flickering" appears, when the user moves from one LDI to the next. Note, that this is especially awkward, when the LDI is viewed in stereo mode and each eye sees an image generated from a different LDI patch. On the other hand, this problem does not occur, when one large LDI is utilized. Using a set of small LDIs, however, provides much better performance than utilizing one large LDI for the whole hemisphere. As stated above, since each two adjacent LDIs contain common depth images, the switching from one LDI to the other is imperceptible for the viewer.

The offscreen rendering applied for all 29 camera positions (8 patches $\times$ 3 cameras and one patch with 5 cameras) can be a very time consuming task. Therefore, initially we compute only the LDI patch covering the current view-direction position. Afterwards, a function is scheduled, which is executed each time the application is idle. Each time the function is executed, the LDI for one patch is computed. If, however, the viewpoint of the user changes, the application computes the needed LDI immediately.

5.4 The LDI Splatter

During the rendering of an LDI onto the window frozen in space, we first determine the position and the angle of the window (LDI-projection plane) relative to the viewer. This determines which LDI of the multi LDI has to be rendered. The current camera position and the projection plane position are used to compute the projective transformation, which maps the LDI pixels onto the projection window. The pixels of the LDI are splatted using a software renderer. We use McMillans's ordering algorithm [McMil95] to access the single pixels in such a way, that they can be successively drawn without applying an explicit depth test. In contrast to Shade et al. [Shade98], however, we compute the splat size for each pixel individually. This increases the precision and avoids visual artefacts caused by the coarse splat size approximation. The resulting picture is used as a texture, which is mapped onto the projection plane using the graphics hardware.

Since the Studierstube is based on Open Inventor, we implemented the LDI-splatter as an Inventor class derived from the regular 2D texture class. Each time an instance of this class is traversed, a new view-dependent texture is generated and mapped onto the given window in space. Thus, it creates the illusion of viewing a true 3D world behind the LDI's projection plane.

5.5 Adding and Removing Objects in an LDI

The scene frozen on the LDI contains all details of the latter, being present when the acquisition was started. However, depending on the application area, updates of the LDI might be desirable. For instance, during the construction of a virtual environment a window may be frozen to show a given invisible area, such that the added new objects are also displayed in this window.

This concept can be easily adapted to the LDI data presentation. When an object is created and it is visible in the LDI window, we can render the object as a geometry object. For this, we need the depth information of the pixels in the final splatted image. Fortunately, the LDI structure presented above can be easily extended to provide this capability. Thus, a mixed rendering including geometry and LDI data can be performed.

6 Application

The LDI-technique described above is integrated in an application for visualization of digital elevation models, urban planning, and terrain exploration. Due to the (almost unlimited) size of the displayed data, this application provides a good test bed for the proposed techniques. Furthermore, the navigation in such large scenes is also a challenging task, which we addressed with the described extension of the WIM-metaphor.

The displayed terrain we used in our system has a size of approximately 12x12 kilometers. The images in Figures 1(b) and Figure 4 showing the map tool were made with this application. The snapshot images are made with the same application, however, on a small, scaled up part of the terrain.

7 Results

In order to evaluate the performance of the proposed technique, we compute the times for the geometry rendering of three scenes with $126'790$, $264'247$, and $630'799$ triangles respectively. Afterwards, we compare the times for rendering various number of snapshot windows represented as geometry data and as multi LDI data (Figure 6). The number of windows varies from none, which means that only the surrounding world is rendered (as geometry), to 4 windows. The measurements are performed on a Pentium III 733Mhz machine (1Gb main memory)

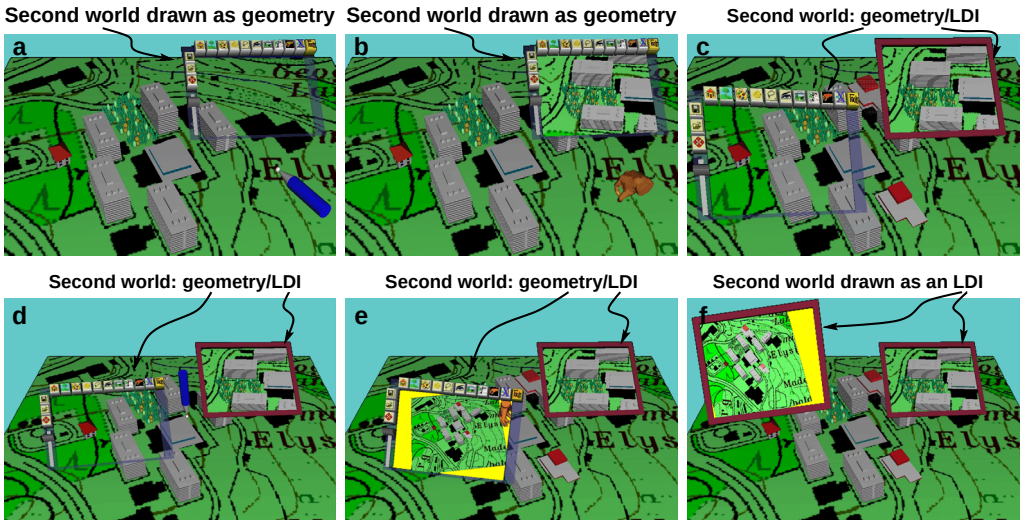
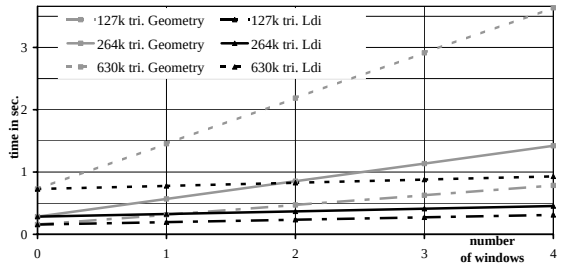


Figure 5: The images (a)-(c) show the snapshot tool: (a) The scene surrounding the user and the one seen through the second viewport are aligned. (b) The scene seen through the pip is grabbed and rotated. (c) the manipulated scene is frozen in an LDI window. (d) Again the new world is aligned with the main world. (e) The second world is manipulated. (f) The manipulated world is frozen in a new LDI-window.



	127k triangles rendered as		264k triangles rendered as		630k triangles rendered as	
Windows	Geometry	Ldi	Geometry	Ldi	Geometry	Ldi
0	0.1567	0.1567	0.2838	0.2838	0.72708	0.72708
1	0.3136	0.1953	0.56788	0.3261	1.45416	0.77736
2	0.4708	0.2341	0.85276	0.3687	2.18624	0.82796
3	0.6272	0.2731	1.1355	0.4111	2.91304	0.87896
4	0.7844	0.3121	1.421	0.4541	3.636	0.93096

Figure 6: Comparison of the geometry rendered snapshot tool and the multi LDI-based rendering.

with NVidia Ge-Force 2 GTS graphics hardware (64Mb texture memory).

The window with the LDI containing the scene was positioned in such a way, that the required resolution of the multi LDI was not more than 256×256 pixels. This is quite large when we consider, that the resolution of the Virtual Table we use in stereo mode is 1024×768. After offscreen-rendering a depth image, the time required for inserting it in an LDI-patch was on average 0.13 seconds. Since each pixel is inserted in a depth-sorted order, this average time varies depending on the number of the pixels, which are already stored in the LDI.

More interesting and important is the rendering time for the LDI, when applied in the proposed tools. Depending on the viewing angle and the distance to the current viewpoint, this time may vary as well. The values for the multi LDI rendering shown in Figure 6 are "worst case" values. They include the time for computing a texture with the given resolution (splatting) and mapping it on the pip.

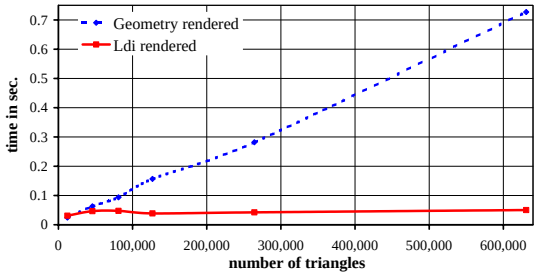


Figure 7: Rendering time for scenes with various complexities, rendered as geometry and as multi LDI data.

Figures 6 and 7 clearly show, that the proposed technique is especially useful, when several views are used during exploration of the scene, and when it contains complex geometry. For instance, let us consider a case in which there are two simultaneous views of the scene and the regular surrounding scene, i.e. the scene is rendered three times per frame. Our scene with $126'790$ triangles requires approximately 0.47 seconds per frame using the standard geometry rendering pipeline. Applying our approach, this

cost is reduced to rendering the scene as geometry once and rendering two multi LDIs. This results in 0.23 seconds at all. In other words, the rendering time was reduced by more than 50%, which means that the frame rate was more than doubled.

8 Conclusions and Future Work

The only expensive part of our approach is the offscreen rendering. Thus, the next step in our implementation will be the parallelization of the LDI acquisition, which in turn will reduce the latency of the system during the offscreen rendering.

To summarize, in this paper we proposed a set of rendering and interaction techniques for VR-applications. First, we presented the *multi LDI*-concept for covering a hemisphere with small LDI-patches. Such a data structure can be rendered significantly faster than a single large LDI. Afterwards, we described a modification of the WIM-technique, based on a pre-computed multi LDI.

We discussed in detail the implementation of the above techniques. Then, we compared the achieved results with the regular geometry rendering, and briefly presented the application and the setup, the techniques are integrated in. In our opinion, the proposed multi LDI approach is a promising modification of the standard WIM and snapshot tools, considering the growing amount of data managed in virtual worlds nowadays.

REFERENCES

[Aliag97] Daniel G. Aliaga and Anselmo A. Lastra. Architectural walkthroughs using portal textures. In *IEEE Visualization '97*, pages 355–362. 1997.

[Aliag99] Daniel G. Aliaga and Anselmo A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *SIGGRAPH 99 Conference Proceedings*, pages 307–316, August8–13 1999.

[Bier86] Eric A. Bier. Skitters and jacks: Interactive 3D positioning tools. In *Proc. 1986 ACM Workshop on Interactive 3D Graphics*, pages 183–196, October 1986.

[Bier93] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose. Toolglass and magic lenses: The see-through interface. In *SIGGRAPH 93 Conference Proceedings*, pages 73–80, 1993.

[Darke93] Rudolph P. Darken and John L. Sibert. A toolset for navigation in virtual environments. In *Proceedings of the ACM Symposium on UIST*, pages 157–165, 1993.

[Decor99] Xavier Decoret, François Sillion, Gernot Schaufler, and Julie Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum*, 18(3):61–73, September 1999.

[McMil95] Leonard McMillan. A list-priority rendering algorithm for redisplaying projected surfaces. Technical Report TR95-005, University of North Carolina - Chapel Hill, February 14 1995.

[Mine95] Mark Raymond Mine. Virtual environment interaction techniques. Technical Report TR95-018, University of North Carolina - Chapel Hill, May 4 1995.

[Olive99] Manuel M. Oliveira and Gary Bishop. Image-based objects. In *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 191–198. 1999

[Pausc95] Randy Pausch, Tommy Burnette, Dan Brockway, and Michael E. Weiblen. Navigation and locomotion in virtual worlds via flight into Hand-Held miniatures. In *SIGGRAPH 95 Conference Proceedings*, pages 399–400. August 1995.

[Raffe97] Matthew M. Rafferty, Daniel G. Aliaga, and Anselmo A. Lastra. 3D image warping in architectural walkthroughs. Technical Report TR97-019, University of North Carolina - Chapel Hill, September 03 1997.

[Raffe98] Matthew M. Rafferty, Daniel G. Aliaga, Voicu Popescu, and Anselmo A. Lastra. Images for accelerating architectural walkthroughs. *IEEE Computer Graphics & Applications*, 18(6), November – December 1998.

[Schau98] Gernot Schaufler. Per-object image warping with layered impostors. In *Rendering Techniques '98*, Eurographics, pages 145–156. Springer-Verlag Wien New York, 1998.

[Schma96] Dieter Schmalstieg, Anton L. Fuhrmann, Michael Gervautz, and Zsolt Szalavári. 'Studierstube' - An Environment for Collaboration in Augmented Reality'. In *Proceedings of Collaborative Virtual Environments '96*, 1996.

[Schma99] Dieter Schmalstieg, L. Miguel Encarnação, and Zsolt Szalavári. Using transparent props for interaction with the virtual table (color plate S. 232). In *Proceedings of the Conference on the 1999 Symposium on Interactive 3D Graphics*, pages 147–154, April 26–28 1999.

[Shade98] Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *SIGGRAPH 98 Conference Proceedings*, pages 231–242. July 1998.

[Stoak95] Richard Stoakley, Matthew J. Conway, and Randy Pausch. Virtual reality on a WIM: Interactive worlds in miniature. In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, pages 265–272, 1995.

[Stoev01] Stanislav L. Stoev, Dieter Schmalstieg, and Wolfgang Straßer. Two-handed through-the-lens-techniques for navigation in virtual environments. In *Proceedings of the Eurographics Workshop on Virtual Environments*, 16-18 May 2001.

[Szala97] Zs. Szalavári and M. Gervautz. The personal interaction panel - A two handed interface for augmented reality. *Computer Graphics Forum (Proceedings of EUROGRAPHICS'97)*, 16(3):335–346, 1997.

[Ware90] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics* pages 175–183, 1990.

[Wimme99] Michael Wimmer, Markus Giegl, and Dieter Schmalstieg. Fast walkthroughs with image caches and ray casting. *Computers and Graphics*, 23(6):831–838, December 1999.