Real-Time Rendering of 3D Magic Lenses having arbitrary convex Shapes

Timo Ropinski
Institut für Informatik, WWU Münster
Einsteinstrasse 62
D-48149 Münster, Germany
ropinski@math.uni-muenster.de

Klaus Hinrichs
Institut für Informatik, WWU Münster
Einsteinstrasse 62
D-48149 Münster, Germany
khh@uni-muenster.de

ABSTRACT

We present a real-time algorithm for rendering volumetric 3D *Magic Lenses*™ having arbitrary convex shapes. During fragment processing the algorithm performs a second depth test using a shadow map. Exploiting the second depth test we are able to classify each fragment, with respect to its position relative to the lens volume. Using this classification we first render the geometry behind the lens volume, then the geometry intersecting the lens volume using a different visual appearance and finally the parts in front of the lens volume. Regardless of the shape of the lens volume just two additional rendering passes are needed. Furthermore there are no limitations to the choice of visual appearance used to enhance expressiveness of the virtual world. We will describe theoretical and practical aspects of the algorithm and our implementation, which is accelerated by current graphics hardware.

Keywords

3D Magic Lenses, 3D Exploration, Toolglass, See-Through Tools

1. INTRODUCTION

User centered exploration of 3D environments becomes more and more important with the advancing development of 3D graphics processing units (GPU). Due to the availability of current graphics hardware at low prices, such hardware is now widely installed in desktop computers giving more and more users access to 3D graphics applications. Therefore user centered visualization techniques are needed, which can be implemented on off-the-shelf graphics hardware.

The magic lens metaphor has been proved to be a powerful tool for exploring the virtual world [Sto02]. But up to now mostly the 2D through-the-lens metaphor is used to explore and manipulate 2D datasets in 2D graphics applications, because no technique is known which is capable of rendering arbitrary volumetric magic lenses in real-time (i.e. with interactive frame rates). In this paper we present an algorithm for rendering volumetric magic lenses having arbitrary convex shapes, taking advantage of the features provided by off-the-shelf graphics hardware. Our algorithm is a multipass-rendering algorithm which performs a second depth test using a shadow map. The algorithm needs only two additional rendering passes and can therefore be used easily in existing 3D graphics applications.

Journal of WSCG, Vol.12, No.1-3, ISSN 1213-6972
WSCG'2004, February 2-6, 2004, Plzen, Czech Republic.

Most of the concepts developed for 2D magic lenses can be applied to our representation of a volumetric magic lens. Integrating our approach into a high-level rendering toolkit gives the application programmer the ability to modify the visualization as well as the interaction techniques associated with parts of a 3D scene, and hence allows the end user to change the region of interest interactively. Therefore we believe that applying the magic lens metaphor in 3D improves usability while exploring a virtual world.

In our approach the magic lens, which is visually represented by an arbitrary convex glass volume, can be positioned and resized interactively to change the region of interest. This region is visually emphasized by the filter functionality of the lens which can be altered interactively as well. New lenses with new filter functionality can be derived from existing sample implementations.

The idea of applying the magic lens metaphor to virtual environments has been proposed before ([Cig94], [Vie96]), but existing approaches lack

performance and flexibility regarding the shape of the lens. Our algorithm has the following advantages over currently known techniques:

- capability of rendering magic lenses having arbitrary convex shapes,
- the implementation of our algorithm is hardware-accelerated and therefore permits interactive frame rates while rendering arbitrary 3D scenes,
- it can be easily extended to support rendering of more then one magic lens per scene and
- due to its simple structure, the algorithm can be easily embedded in existing graphics applications and toolkits.

This paper proceeds in discussing related works in Section 2. Section 3 introduces the basic idea underlying the algorithm. Section 4 describes our implementation exploiting a second depth test which uses a shadow map. Section 5 discusses some application areas potentially benefiting from the use of magic lenses in 3D. Section 6 gives performance measurements and a brief discussion of our results, and Section 7 concludes the paper with a short outline of future work concerning 3D magic lenses.

2. RELATED WORK

Magic Lenses

The magic lens metaphor has been introduced in 1994 by Bier et al. [Bie94a]. In their work they describe *Toolglass*™ widgets as new interface tools that can appear, as though on a transparent sheet of glass, between an application and a traditional cursor. Toolglass widgets can be positioned with one hand while the other positions the cursor. They may incorporate visual filters, known as magic lenses, which modify the visual appearance of application objects, enhance data of interest or suppress distracting information in the region of interest, which is determined by the shape of the lens. Besides the use in 2D graphics applications Bier et al. describe how to apply the concept of magic lenses and toolglasses for text editing. They also give an example how to apply the concept of 2D planar lenses to 3D virtual environments. Inspired by the initial idea of the magic lens metaphor [Bie94a] several papers followed, covering a taxonomy [Bie94b] and composition of magic lenses [Fox98] as well as several applications of the concept ([Bie97], [Stn94], [Sto02]).

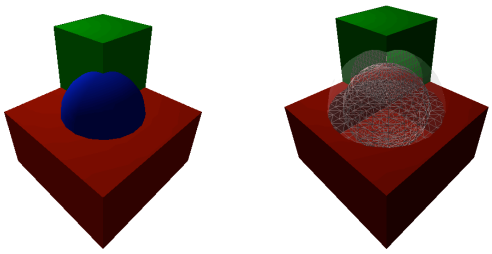
Cignoni et al. [Cig94] were the first to transfer the magic lens metaphor to volumetric lenses. In their work they describe the *MagicSphere* metaphor representing an insight tool for 3D data visualization. As the name implies, the metaphor is limited to a spherical lens volume. Besides this restriction to one usable lens shape the visual appearance lacks due to their analytical approach: In a preprocessing step geometrical elements are classified based on their position relative to the border of the magicsphere. Subsequent rendering requires two passes; one for the geometrical elements lying outside the magicsphere and one for those inside the magicsphere. The elements lying on the border, classified as border elements, are rendered in both of the two passes. When using the magicsphere with a *MultiRes* filter Cignoni et al. obtain a satisfactory visual appearance even for the border elements, which are rendered twice. However, the visual appearances used in the two rendering passes of their *EdgesEmphasizer* filter differ too much and therefore cause visual artifacts near the border of the magicsphere. Similar artifacts have to be expected with other visualization techniques used by the magicsphere metaphor.

Another more general extension of the magic lens metaphor to 3D virtual environments has been presented by Viega et al. [Vie96]. They introduced an algorithm for visualizing volumetric lenses as well as flat lenses in a 3D environment. Their implementation of these concepts exploits SGI Reality Engine hardware support for clipping planes. Since only infinite clipping planes are supported, it takes an extra rendering pass for almost every face of the lens volume. Thus it would be computationally very expensive to render magic lenses having arbitrary shapes. Another disadvantage results from the limited number of clipping planes supported by current graphics devices increasing the number of necessary rendering passes.

A different analytical approach of a similar concept has been used by Idelix Software Inc. [Ide02]. Their Pliable Display Technology 3D (*PDT3D*) avoids object occlusions in 3D virtual environments by analyzing camera and lens parameters and applying corresponding geometric transformations to occluding objects. Thus it is possible to select a region of interest, to which the system provides an occlusion-free view. The major disadvantage of this concept is the modification of the scene structure lying outside the region of interest through geometrical transformations, which leads to a loss of contextual information.

The powerful concept of magic lenses has been deployed in many fields. But due to the computationally expensive realization of volumetric magic lenses, leading to a limitation of lens shapes and overhead in rendering time, mostly 2D magic lenses have been applied so far. The concept has been used for example in Kai's Power Tools [Vie96], a collection of innovative filters for 2D image processing, as well as in Macromedia Freehand™, a tool for creating illustrations. But also users of 3D applications can benefit from the concept of magic lenses. For example in figure 1 the underlying geometry is revealed by the

applied wireframe lens, giving a better overview of object composition without loss of contextual information.



(a) (b)

Figure 1: Sample scene without magic lens (a). Magic lens reveals the inside structure (b).

Shadow Mapping

Shadow mapping has been introduced by Williams [Wil78] in 1978. To display shadows in real-time, the scene is rendered in a preprocessing step using the position of the light source as a viewpoint. The resulting depth buffer information, which represents the surfaces visible from the light source, is stored in a shadow map sometimes referred to as depth texture. During the main rendering pass, each point is transformed into the camera coordinate system. This way it is possible to compare the distance of any point to the light source with that of the closest point to the light source. Points that are farther from the light source than the closest one lie in shadow and are therefore rendered less illuminated.

Because shadow mapping is a widely accepted real-time shadowing technique, it is accelerated by current graphics hardware.

Depth Peeling

In this subsection we will give a brief description of the depth peeling technique [Eve02], which uses shadow maps to enable a second depth test and therefore benefits from hardware-acceleration.

Everitt describes the use of depth peeling to achieve order independent transparency. The idea is to *peel away* layers of the scene from front to back on a fragment level and render those layers in inverted order. To obtain a specific fragment layer it is necessary to utilize an additional depth test to get all fragments contained in that layer.

Considered as a multipass rendering technique depth peeling works as follows. The first pass uses the standard depth test and therefore writes the depth- and RGBA-values of the nearest fragment to the respective buffers. The depth buffer information generated by this pass is copied to a shadow map, which serves as the depth buffer for the second depth test performed in the subsequent rendering pass. Iterating this process over the necessary n passes gives the ability to *peel away* and obtain the n-th nearest fragment layers.

Using this technique it is possible to get n layers deeper into a scene with n passes, whereby in each pass the shadow map, generated in the preceding pass, serves as the depth buffer for the second depth test. Projective texture mapping [Eve01] is used to project the shadow map onto the scene aligned along the eye view image plane. This projection makes sure that fragments of the shadow map overlay corresponding fragments (i.e. with same window coordinates) of the depth buffer. To compare the depth values of the corresponding fragments the shadow comparison function is used [Pau02b], which assigns an alpha value to the resulting fragment based on its depth value. Finally the alpha test discards fragments as a result of their alpha values.

The read-only limitation of the shadow map is no restriction, because the depth information for the n-th pass is created in the (n-1)-th pass and therefore it is not necessary to write into both depth buffers in one rendering pass.

3. ALGORITHM

Our image-based algorithm utilizes functionality similar to shadow mapping to achieve an effect similar to depth peeling while fragment processing.

For explaining the basic idea of the algorithm consider the following classification (see figure 2) into fragments lying

(a) behind the lens,
(b) inside the lens,
(c) in front of the lens and
(d) the remaining fragments.

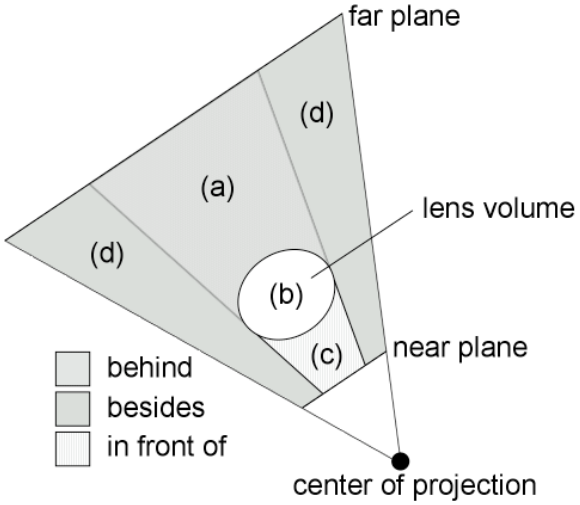


Figure 2. Subdivision of the view frustum into three sections (behind, inside, in front of) depending on camera parameters.

The first rendering pass uses two depth tests to exclude fragments lying inside (b) and in front of (c) the lens from rendering. The second rendering pass also uses two depth tests to exclude fragments lying outside the lens ((a), (c), (d)) whereas the third pass only needs one depth test to exclude fragments inside (b) and behind (a) the lens.

Visible fragments belonging to (d) are rendered in the first and third pass.

The order of the three rendering passes is relevant because some parts of the scene lying inside or in front of the magic lens could be semi-transparent. Therefore it is important to render the parts ((a), (b), (c)) in a back to front order to preserve a correct image in case of semi-transparency, like it is done during depth peeling.

Since the algorithm works on fragment level regardless of the shape of the lens only two additional rendering passes are needed. So unlike the approach of Viega et al. [Vie96] the complexity of the lens shape does not affect the rendering complexity.

4. IMPLEMENTATION

The presented algorithm has been realized in C++ using OpenGL as rendering library. Furthermore it has been integrated into a high-level graphics framework called *VRS* (Virtual Rendering System) [Döl02] exploiting the integrated scenegraph structure.
Our algorithm is a straight-forward multipass-rendering algorithm. Like in the depth peeling technique, we use the shadow comparison function [Pau02b] along with the alpha test to perform the second depth test. The alternative to use a fragment program [Lip03] to discard fragments due to their alpha value has been implemented as well.

Assuming that it is possible to use two depth buffers with separate configurable depth tests, the complete algorithm is given by the following pseudo code.

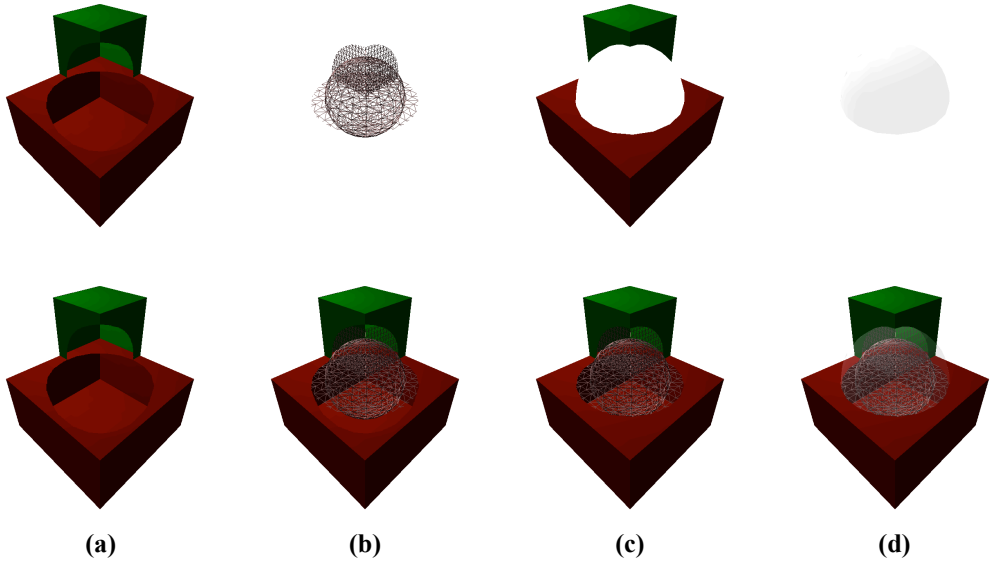
```
// pass 1: render geometry
// behind the lens
clearDepthBuffer1 ( 0.0 );
setDepthTest1 ( GREATER );
clearDepthBuffer2 ( 1.0 );
setDepthTest2 ( LESS );
actDepthBuffer ( DepthBuffer1 );
renderLens ( );
actDepthBuffer ( DepthBuffer2 );
renderScene ( NORMAL );
// pass 2: render geometry
// intersecting the lens
clearDepthBuffer1 ( 1.0 );
setDepthTest1 ( LESS );
clearDepthBuffer2 ( 0.0 );
setDepthTest2 ( GREATER );
actDepthBuffer ( DepthBuffer1 );
renderLens ( );
actDepthBuffer ( DepthBuffer2 );
setDepthTest1 ( ALWAYS );
renderLens ( );
setDepthTest1 ( GREATER );
setDepthTest2 ( LESS );
renderScene ( LENS_STYLE);
// pass 3: render geometry
// in front of the lens
clearDepthBuffer1 ( 1.0 );
setDepthTest1 ( LESS );
setDepthTest2 ( ALWAYS );
actDepthBuffer ( DepthBuffer1 );
renderLens ( );
renderScene ( NORMAL );
```

`actDepthBuffer()` activates one of the two depth buffers `DepthBuffer1` and `DepthBuffer2` for writing. `setDepthTest1()` and `setDepthTest2()` configure the corresponding depth test. `clearDepthBuffer1()` and `clearDepthBuffer2()` overwrite data in the corresponding depth buffer with the assigned value.

`renderScene()` executes the instructions for rendering the scene data, `renderLens()` the instructions for rendering the lens geometry. `renderScene()` updates both, the color buffer and the depth buffer, whereas `renderLens()` writes only to the depth buffer. `renderScene()` expects either the parameter `NORMAL` or `LENS_STYLE` to determine which viusal appearance to use for rendering.

The structure of the algorithm indicated by the integrated comments corresponds to the three needed rendering passes. The result of each of the three rendering passes of the algorithm is shown in figure 3 (a-c) in the top row, using a spherical wireframe lens to reveal insights into the scene geometry. The bottom row shows the accumulated result after each rendering pass (hence the two images shown in figure 3 (a) are the same). Figure 3 (d) shows the visualization of the lens by a semi-transparent lens volume. In order to prevent the lens from occluding scene geometry this requires an additional fourth rendering pass to update depth buffer information before rendering the lens.

So far we have assumed the existence of a second depth buffer with an independently configurable depth test. Unfortunately current graphics systems do not support a second depth test. However, it is possible to simulate a second depth buffer with a separately configurable depth test using a shadow map. Although this simulated depth buffer does not allow writing, this poses no limitation to our algorithm because the lens geometry is rendered in a separate rendering pass.

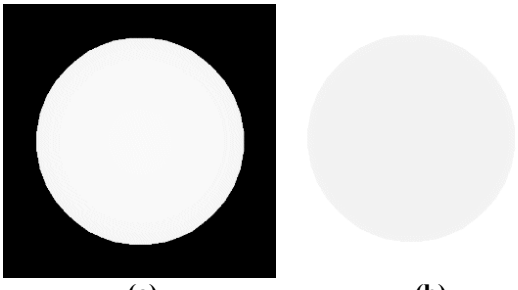


(a) (b) (c) (d)

Figure 3. Content of the color buffer after each of the four rendering passes (a-d). Top row: resulting color information after each pass; bottom row: accumulated images.

To perform the second depth test in the first and second rendering pass, two different shadow maps are needed. The shadow map in the first pass represents the back facing polygons of the lens volume, and the map in the second pass the front facing polygons. Each of these shadow maps stores the depth buffer information obtained by rendering the lens geometry to an offscreen canvas.

Two examples of generated shadow maps are shown in figure 4, where brighter pixels represent greater depth values. The shadow map in figure 4 (a) which contains the depth information obtained by rendering the back of a spherical lens is used in the first pass to render the fragments lying behind the lens volume (a). The map in figure 4 (b) which corresponds to the front of the same lens is used in the second pass for rendering the fragments inside the lens volume (b).



(a) (b)

Figure 4. Depth information of (a) the back and (b) the front of the lens.

Each of the two shadow maps generated in the first and second rendering pass is aligned with the scene by projective texture mapping [Eve01]. This ensures that the depth value of the current fragment can be compared by the shadow comparison function [Pau02b] to the depth value of the corresponding shadow map texel. The alpha value of the current fragment is set depending on the result of this comparison. After evaluating the shadow comparison function the alpha test is used to discard fragments not needed in the current rendering pass. Thus we can assure that only fragments needed in the current rendering pass are written to the frame buffer.

5. APPLICATION

We have implemented two kinds of volumetric magic lenses: The position of a *camera lens* is fixed relative to the camera, i.e. when the camera moves then the lens also moves. In contrast a *scene lens* can be positioned anywhere in the virtual environment. Camera lenses can be used similar to the PDT3D [Ide02] to explore a scene with mostly dense data. A potential application area would be the exploration of subsurfaces where magic lenses can reveal a better insight by displaying data next to the camera differently, for example semi-transparently. Scene lenses can be used to assist 3D modeling. Many other application areas can benefit by exploiting the metaphor of volumetric magic lenses:

- wireframe lenses are capable of displaying vertex details for parts of the scene without modifying the global view, which would lead to a loss of context information (figure 5),
- eraser lenses (figure 8) can reveal occluded parts of the scene by removing partial or complete information from regions intersecting the lens,
- texture lenses can assist in exploring terrain data, by replacing or modifying the texture of objects. These 3D lenses are a generalization of the 2D texture lenses introduced in [Döl00].

Furthermore the concept of volumetric magic lenses has been used to show different levels of detail for a model [Cig94] and enhance flow visualization [Fuh98].

Our technique for rendering volumetric magic lenses can benefit from the scenegraph concept used in most recent high-level rendering systems ([Str92], [Döl02], [Rei02], [Sel02]). The algorithm as described above evaluates the whole scene geometry in each of the three rendering passes. However, by combining a scenegraph with space partitioning the amount of data to be processed in each rendering pass can be reduced. If bounding box extensions are provided in each scene node as in Java3D [Sel02] intersections with the lens volume can be detected more efficiently. This leads to a reduction of rendering data, which is significant for the second rendering pass since usually magic lenses are very small related to the virtual environment.

6. DISCUSSION

Our algorithm has many advantages compared to the clipping plane based approach introduced by Viega et al. [Vie96]. Since the algorithm works on a fragment level the number of rendering passes does not increase with the complexity of the lens volume, which makes it possible to use arbitrary convex shapes as lenses. Thus it is possible to use e.g. quadric shaped lenses as shown in figure 5 without generating visual artifacts, which arise for example when combining the magicsphere metaphor [Cig94] with common rendering styles.

shadow map resolution	fps
no lens used	106,50
128x128	44,72
256x256	44,39
512x512	44,10
1024x1024	43,12

Table 1. Frame rates achieved by rendering the scene shown in figure 5.

Unlike Viega's [Vie96] approach the performance of our technique does not depend on the shape of the lens volume. Even the offscreen rendering to obtain the shadow maps does not result in a noteworthy rendering overhead since a hardware-accelerated pixel buffer is used and only the lens geometry has to be rendered without shading. Furthermore it would be possible to use render-to-texture functionality to accelerate offscreen rendering.

In order to determine the rendering performance of our algorithm, we used the scene in figure 5 for measuring frame rates. Table 1 contains the frame rates achieved by rendering the scene with no lens and with the application of a spherical wireframe lens using different shadow map resolutions. We used a Pentium 4 2,66 GHz system, running Windows XP Professional, equipped with 1 GB RAM and an ATI Radeon 9800 Pro graphics card with 128 MB RAM.

Figure 5. Application of a quadric wireframe lens to an arbitrary 3D model.

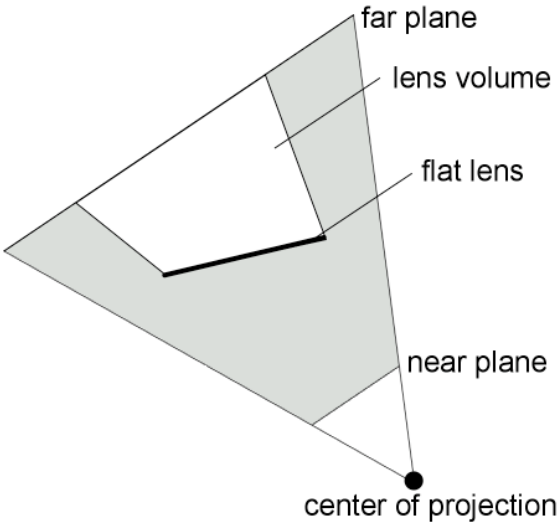


Figure 6. Definition of two sections (lens, outside) depending on the camera parameters using a 2D flat lens.

It is obvious, that our technique is also capable of rendering convex flat lenses introduced by Viega et al. [Vie96]. In our approach, a flat lens is just a special case of a volumetric 3D lens. It can be placed anywhere in the scene and its sphere of influence, called the lens frustum, ranges from its 2D shape to the far clipping plane. This divides the scene into two regions and therefore decreases the number of additional rendering passes to one (see figure 6). Thus it is possible to render flat lenses with our algorithm

using two passes, similar to the procedure for rendering volumetric lenses.

Another advantage of our approach is its sparse need of resources. It only uses one texture to perform the second depth test. Thus it is fairly easy to integrate the concept of magic lenses with other multipass rendering techniques like shadow or reflection generation.

A critical aspect is formed by volumetric lenses intersecting the near or the far clipping plane. While rendering the parts of a scene intersecting the lens volume our algorithm requires the lens volume to have a front- and a back-facing region, which form the lens volume. Lenses intersecting the near or the far clipping plane do not have an intuitively defined front- or back-facing region leading to an undefined lens volume (see figure 7). It is obvious that the problem is even worse concerning lenses intersecting both the near and the far-clipping plane. Although this problem has not been solved yet, it can be avoided by using camera lenses (see section 5) positioned between the near and the far clipping plane without intersecting them.

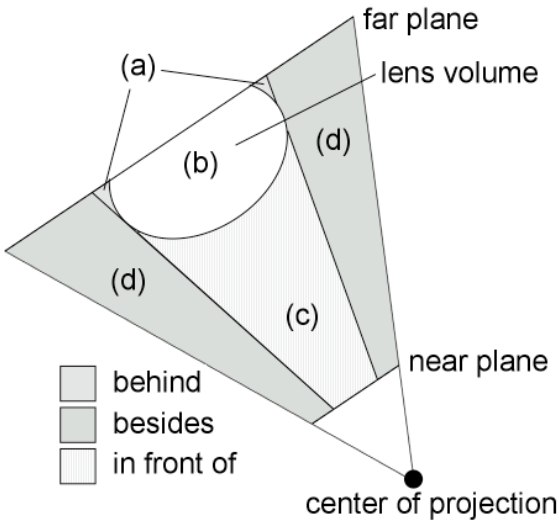


Figure 7. Lens intersecting the far clipping plane.

7. CONCLUSION AND FUTURE WORK

We have presented an algorithm for real-time rendering of volumetric magic lenses having arbitrary convex shapes, which is fully hardware-accelerated.

The concept of magic lenses supports the combination of different visualization appearances in one scene, giving the user a better insight regarding his region of interest. We have presented several applications of volumetric magic lenses. We will investigate more application areas and develop different kinds of magic lenses to use in 3D virtual environments.

Furthermore we are working on an approach for rendering more than one magic lens per scene, which is closely related to the combination of visualization techniques associated with overlapping magic lenses. Due to the nature of volumetric magic lenses their combination in 3D often leads to non-convex regions having their own visualization techniques, which we are implementing right now. Using this extension we are confident to be able to render more general non-convex magic lenses as well.

8. ACKNOWLEDGEMENTS

Magic Lenses™ and Toolglasses™ are Trademarks of the Xerox Corporation.

9. REFERENCES

[Bie94a] E. A. Bier, M. C. Stone, K. Pier, W. Buxton and T. DeRose: Toolglass and Magic Lenses: The See-Through Interface. In Proceedings of SIGGRAPH'93, pages 73–80. ACM Press, 1993.

[Bie94b] E. A. Bier, M. C. Stone, K. Fishkin, W. Buxton and T. Baudel: A Taxonomy of See-Through Tools. In Proceedings of CHI'94, pages 358–364, Boston, April 1994.

[Bie97] E. A. Bier, M. C. Stone and K. Pier: Enhanced Illustration using Magic Lens Filters. Xerox Palo Alto Research Center 1997.

[Cig94] P. Cignoni, C. Montani, and R. Scopigno: MagicSphere: An insight tool for 3d data visualization. In Eurographics'94, pages 317–328, 1994.

[Döl00] J. Döllner, K. Baumann and K. Hinrichs: Texturing techniques for terrain visualization. In T. Ertl, B. Hamann, and A. Varshney, editors, Proceedings Visualization 2000, pages 227--234. IEEE Computer Society Technical Committee on Computer Graphics, 2000.

[Döl02] J. Döllner and K. Hinrichs: A Generic Rendering System, IEEE Transactions on Visualization and Computer Graphics '02, pages 99-118, 2002.

[Eve01] C. Everitt: Projective Texture Mapping. White paper, NVidia Corporation, 2001. http://developer.nvidia.com/attach/1455 (PDF format)

[Eve02] C. Everitt: Interactive Order-Independent Transparency. White paper, NVidia Corporation, 2002. http://developer.nvidia.com/attach/1451 (PDF format)

[Fox98] D. Fox: Composing Magic Lenses. Proceedings of ACM CHI'98, pages 519-525, 1998.

[Fuh98] A. Fuhrmann and E. Gröller: Real-time techniques for 3D flow visualization. IEEE Visualization '98, pages 305-312, 1998.

[Ide02] Pliable Display Technology 3D: White paper, IDELIX Software Incorporation, 2002.

[Lip03] B. Lipchak (Ed.): NVidia OpenGL Extension Specifications. NVidia Corporation, 2003. http://www.nvidia.com/dev_content/nvopenglspecs/GL_ARB_fragment_program.txt

[Pau02a] B. Paul (Ed.): NVidia OpenGL Extension Specifications. NVidia Corporation, 2002. http://www.nvidia.com/dev_content/nvopenglspecs/GL_ARB_depth_texture.txt

[Pau02b] B. Paul (Ed.): NVidia OpenGL Extension Specifications. NVidia Corporation, 2002. http://www.nvidia.com/dev_content/nvopenglspecs/GL_ARB_shadow.txt

[Rei02] D. Reiners, G. Vo and J. Behr. OpenSG: Basic concepts. In 1. OpenSG Symposium OpenSG 2002, 2002.

[Sel02] D. Selman: Java 3D Programming. Manning Greenwich, 2002.

[Sto02] S. Stoev, D. Schmalstieg and W. Straßer: The Through-The-Lens Metaphor: Taxonomy and Application. Proceedings of the IEEE VR'02, pages 285-286, 2002.

[Stn94] M. C. Stone, K. Fishkin and E. A. Bier, "The Movable Filter as a User Interface Tool." In Human Factors in Computing Systems: Proceedings of the CHI '94 Conference. New York: ACM, 1994.

[Str92] P. Strauss and R. Carey: An object oriented 3D graphics toolkit. In Proceedings SIGGRAPH '92, pages 341–347, 1992.

[Vie96] J. Viega, M. Conway, G. Williams and R. Pausch: 3D Magic Lenses. Proceedings of ACM UIST'96, pages 51-58, 1996.

[Wil78] L. Williams: Casting curved shadows on curved surfaces. Proceedings of SIGGRAPH'78, pages 270-274, 1978.

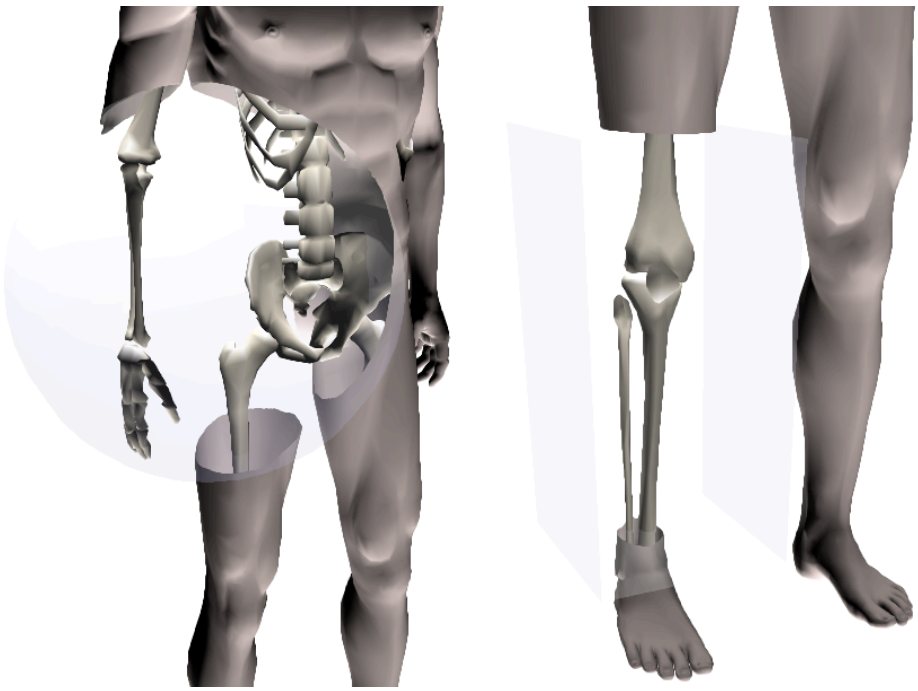


Figure 8: Applications of eraser lenses.