LaMar_etal_2001

Eric LaMar, Bernd Hamann, Kenneth I. Joy,
"A Magnification Lens for Interactive Volume Visualization",
in "Proceedings of Pacific Graphics 2001",
pp 223-233, October 16-18, 2000

A Magnification Lens for Interactive Volume Visualization

Eric LaMar
Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory
lamar1@llnl.gov

Bernd Hamann and Kenneth I. Joy
Center for Image Processing and Integrated Computing,
Dept. of Computer Science, University of California, Davis
{hamann,joy}@cs.ucdavis.edu

Abstract

Volume visualization of large data sets suffers from the same problem that many other visualization modalities suffer from: either one can visualize the entire data set and loose small details or visualize a small region and loose the context. In this paper, we present a magnification lens technique for volume visualization. While the notion of a magnification-lens is not new, and other techniques attempt to simulate the physical properties of a magnifying lens, our contribution is in developing a magnification lens that is fast, can be implemented using a fairly small software overhead, and has a natural, intuitive appearance. The issue with magnification lens is the border, or transition, region. The lens center and exterior have a constant zoom factor, and are simple to render. It is the border region that blends between the external and interior magnification, and has a non-constant magnification. We use the "perspective-correct textures" capability, available in most current graphics systems, to produce a lens with a tessellated border region that approximates linear compression with respect to the radius of the magnification lens. We discuss how a "cubic" border can mitigate the discontinuities resulting from the use of a linear function, without significant performance loss. We discuss various issues concerning development of a three-dimensional magnification lens.

1 Introduction

To properly understand data, a person must be able to see the details while being able to place these details in the larger context of the total data. Data sets are becoming so large that rendering them to standard display devices results in severe under-sampling. For example, it is not possible to view 2000^2 pixels on a $1280*1024$ pixel display: the image resolution is simply too large. In this case, a user can use one of several non-intuitive schemes: closely examine, by magnification or perspective zoom, various regions, zooming in and out to determine location. The second scheme is a dual-window design, where one (local) window shows a zoomed region and the second (world) window shows a severely subsampled version of the full data set, with a box glyph showing the location of the zoom window. Both of these techniques are commonly found in paint, picture-editing, and CAD programs. The first option is poor as features may be larger than the zoomed region, or might not be viewed in their entirety, and thus not be recognized by the user. Also, to navigate to another location in the data, the user must zoom out to determine location and zoom back to a new location or move in some direction in hope to find a way to the "correct" location. The second approach is poor as it also has a similar issue with navigation - the user must interpret translation commands in the zoom window to the world window.

In this paper, we will discuss a magnification-

lens zoom technique for volume visualization of very large data sets. We use hardware-accelerated, texture-based volume visualization, an extension of earlier work, see [8, 7, 6]. We have two requirements for our magnification lens. First, it must have a "natural" or intuitive appearance, in the sense that, as a user moves the lens through a data set, the magnification and transformation of the data seems proper. We are not interested in reproducing the properties of a physical magnifying glass - significant other work of human interaction demonstrates that glyphs and other artificial constructs are sufficient, and sometimes, better than any physical or "real-world" transformation. We develop, compare, and contrast square and circular lens bases. We also develop, compare, and contrast performance and visual characteristics of linear and cubic bases for *warp*ing between magnified and un-magnified regions of an image.

We discuss related work in Section 2. We develop the basic zooming texture techniques for two- and three-dimensional data in Section 3 and discuss issues of geometry in Section 4. Performance results are covered in Section 5. Future work is discussed as part of the conclusions in Section 6.

2 Related Work

Sarkar and Brown [10] developed a Global Fish-eye technique for browsing large graphs (of vertices and edges) by allowing local magnification of graph elements. Their magnification basis is the movement of vertices away from some *focal area*.

Sarkar and Brown [11] also introduced a *rubber-sheet-stretching* metaphor for the local magnification of large graphs. They defined *orthogonal* and *polygonal* stretching bases. The former stretches different ranges of each axis independently, and the latter stretches the entire domain as a function of distance and orientation to a set of foci. The user can manipulate the stretching through handles. The downside of the orthogonal approach is that regions that are not in the set of foci are affected. They solve this problem with the *polygonal* basis. While more expensive, a polygonal basis generates a more intuitive magnification.

Bier *et. al.* [1] introduce the *Magic Lens* as a transparent interaction tool to overlay a workspace. The Magic Lens provides two levels of functionality. The first aspect is that the Magic Lens is represented in outline, as compared to opaque pop-up windows or menus, as to not cover or interfere with the workspace. Second, visualization of attributes of the data can be added or removed. This is important with dense or multivariate data, as it is generally not possible to comprehend an image with more than a few dimensions of data shown. For example, a magic lens could have four or six sub lenses, each showing a different set of variables, or the same variables with different representations. The user can switch different views around by repositioning the larger lens such that the smaller lens covers the region in question. While their work does not discuss "in-context" magnification, it is the basis of many other works developing magnification techniques.

Ware and Lewis [13] discussed the "DragMag Image Magnifier," a two-window approach to view large (12500^2 pixel) images. The first window shows the global view, while the second shows a magnified view. The second window appears as a movable glyph in the global view, and the user can navigate through either window. The user cannot modify the location of relative sizes of the two windows and must mentally fuse the local and global views.

Keahey [3] provides a good summary of the issues and techniques of the "Detail-in-Context" problem. He discusses the uses of glyphs, Level-of-Detail control of glyphs, embedded objects and non-linear magnification of a montage of two-dimensional images.

Rauschenbach [9] developed a demand-driven transmission and magnification approach for large color images. He incorporates wavelet decomposition, transmission, and reconstruction. A simple *orthogonal* stretching method is used.

Keahey introduces volume warping in [4] as a method for easing the examination of a medical database with 11-dimensional records. He defines a non-linear magnification using multiple foci in a three-dimensional volume, allowing the examination of clusters of records while providing the larger context. This technique has the downside in the sense that the data is very sparse and glyphs, in the form of regular, volumetric grid lines, must be added to the rendering to show the location and size of the foci.

Kurzion and Yagel [5] use volume (3D) textures and adaptively tesselated proxy geometries to warp and cut open volumetric objects. While

they can produce interesting deformations, the best use of their technique is for a natural "cutting-open-and-peeling" of an object. None of the images contains a large local stretch (thus, magnification) of the volume and they do not discuss the issues of homogeneous-space textures. We can only assume that there will be significant artifacts in the imagery of highly warp or magnified regions.

Our technique differs from earlier techniques in that we are developing a magnifying lens in the context of large, rectilinear data set volume visualization. We use hardware-texture based volume visualization. We are not as interested in developing mathematical formulas for the deformation of space by a magnification locus, as we are in providing a "natural", intuitive magnification lens that can render imagery very quickly on conventional graphics hardware. We pay particular attention to the correct generation of homogeneous-space textures and how this affects the resulting imagery.

3 Textures

3.1 Zooming with Textures

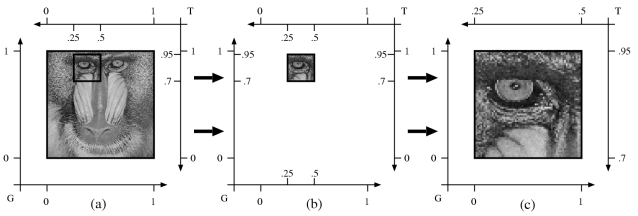
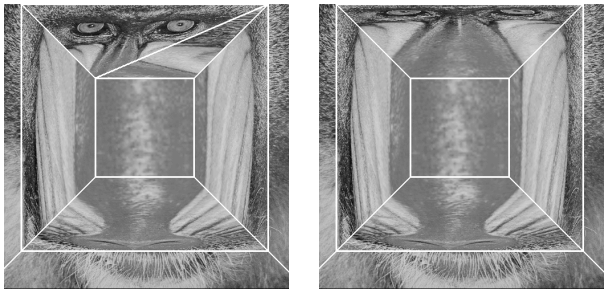


Figure 1. Zooming with textures. Axis G shows geometric space and axis T shows texture space.

Figure 1 illustrates the concept of zooming with textures. Zooming is accomplished by modifying the texture coordinates. The mandrill image is interpreted as a texture defined over a unit domain. In figures 1(a), 1(b), and 1(c), two different domains are shown: the lower-left axis shows the geometric space G, and the upper-right axis shows the texture space T. Figure 1(a) shows the mandrill drawn in geometric and texture space. Figure 1(b) shows a small region and its corresponding geometric and texture coordinates. Figure 1(c) shows this region zoomed by a factor of four by using the geometry from 1(a) and modifying the texture coordinates.

3.2 Warped Surface Textures



(a) 3d TCs (b) 4d TCs

Figure 2. Comparing zoom operations in the top trapezoid: Image (a) shows a zoom operation with texture coordinates (TCs) specified in affine space. Image (b) shows a zoom operation with TCs projected into homogeneous space.

If one wants to increase the scale of a texture, simulating a "zoom-in" operation applied locally, the simplest technique is to move the texture coordinates toward the center of interest. However, if different vertices of a polygon have different zoom factors, a straight modification of texture coordinates to affect a zoom does not work: the hardware is only capable of a linear transformation and the zoom is not linear. Image 2(a) shows the result - this result is clearly not desired. Graphics hardware can only perform linear interpolation of geometric, color, and texture values across a polygon. The authors are not aware of any graphics hardware or graphics specification that allows anything other than linear interpolation. This is due to the fact that linear forward differencing is used during rasterization to calculate values incrementally, from pixel to pixel, across a polygon. Hence, the directional derivative is constant.

Our solution is based on the use of "hardware-implemented, perspective correct textures." This technique, implemented in hardware, projects texture coordinates (TCs) from affine space into homogeneous space, iterates homogeneous texture coordinates during polygon rasterization, and projects the TCs back into affine space for final texture lookup.[1] We note that this tech-

[1]The OpenGL specification defines a texture coordinate by

nique is a special case of projective textures and homogeneous texture coordinates, see [12] for a complete discussion. The center of the images in Figure 2 is magnified by a factor of four, so the texture coordinates associated with the vertices of the center square of the lens are projected into homogeneous space with weight 4. This leads to image 2(b): we have a zoom effect, though with a non-linear compression through the lens border. One last step is necessary to achieve the effect we want.

3.3 Multiple Segments

	Affine	Homogeneous
Lower-Left	$(0, 0, 0, 1)$	$(0, 0, 0, 1)$
Lower-Right	$(1, 0, 0, 1)$	$(1, 0, 0, 1)$
Upper-Left	$(.375, 1, 0, 1)$	$(1.5, 4, 0, 4)$
Upper-Right	$(.625, 1, 0, 1)$	$(2.5, 4, 0, 4)$

Table 1. The affine-space texture coordinates of the *warp* trapezoid of Figure 3 projected into homogeneous space with a weight (zoom) of four.

The last step is to tessellate a polygon into a series of smaller polygons where the new vertices' geometric and texture coordinates linearly interpolate the original polygon's vertex's geometric and texture coordinates. Image set 3(a) shows the original, un-stretched, images with the region to be warped delimited by a red trapezoid and the green arrows showing the direction and degree of the stretch. Each image is stretched at the top by a factor of four, with respect to the bottom edge, which corresponds to $Q = 4$. Table 1 shows the texture coordinates of the vertices of the trapezoid in affine and homogeneous space.

Figure 3 shows one, two, four, eight, and 16 segments for two data sets: a synthetic checker board, and a mandrill image. Column (2) is a synthetic image. We selected it as it demonstrates well the effects of different numbers of segments. Column (4) is the mandrill image and was selected to show how the algorithm affects a known image. Column (1) shows the number of segments used, and column (3) shows the segment boundaries. For simplicity, we refer to the pairs of checker-board and mandrill images in rows 3(a) to 3(f) as "image set" (a) to (f). Each segment is the width of the original polygon and evenly tessellates the original polygon from top to bottom.

With one segment, see image 3(b), we obtain the normal perspective image. With two and four segments, see images 3(c) and 3(d), the existence of the segments is still quite obvious. With eight and 16 segments, see images 3(e) and 3(f), the segments disappear, and we obtain a fairly smooth image. Typically, the lens border will be significantly smaller than these images. However, it is useful to illustrate the effects of the number of segments on the image.

3.4 Border Transitions

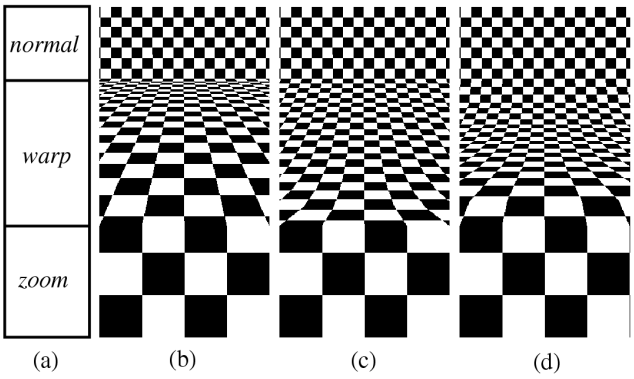


Figure 4. Border examples: *warp*ing the *normal* (top) region into the *zoom*ed (bottom) region. Image (b) is simple; (c) is linear; and (d) is cubic.

We define a magnifying lens to have three parts, see Figure 4(a), from top to bottom: *normal*, *warp*, and *zoom*. Concerning images 4(b) to 4(d), the *normal* and *zoom* regions are the same. What is different in the images is the *warp* region.

Image 4(b) shows a single segment. The image is highly compressed near the *normal-warp* border. In this region, small details can get lost, making it difficult to navigate to them. Also, the discontinuity across the *normal-warp* border is, visually, very strong.

four scalars, or (S, T, R, Q). Thus, affine space corresponds to $(S, T, R, 1)$, and to "project" this value into homogeneous space by a "weight" Q results in $(S \times Q, T \times Q, R \times Q, Q)$. Homogeneous space is equivalent to (S, T, R, Q), and the projection to affine space is $\left(\frac{S}{Q}, \frac{T}{Q}, \frac{R}{Q}, 1\right)$. When using textures of lower dimension, the higher-order affine coordinates are ignored.

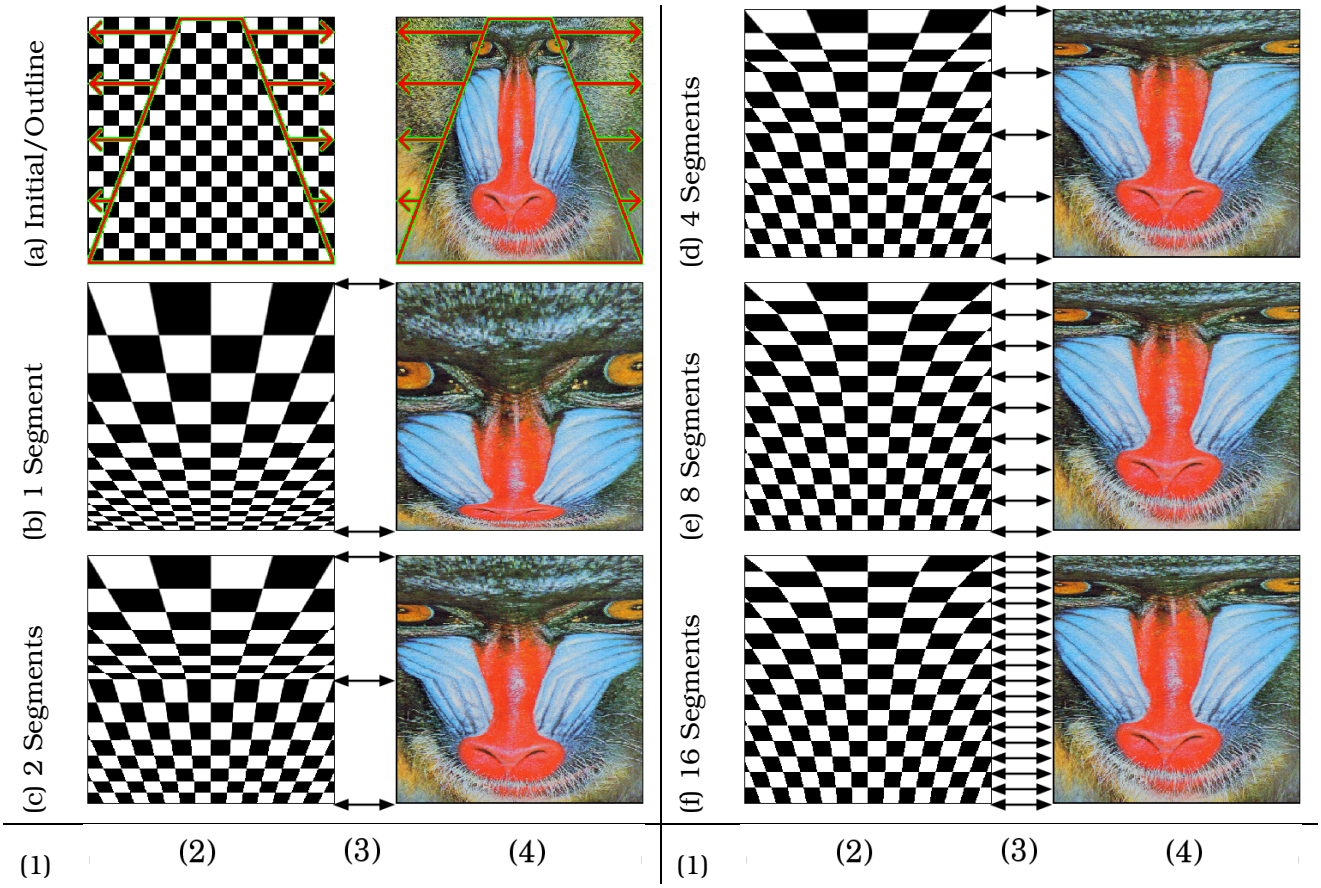


Figure 3. Multiple Segments. The top border is stretched by a factor of four. The double-headed arrows show segment boundaries.

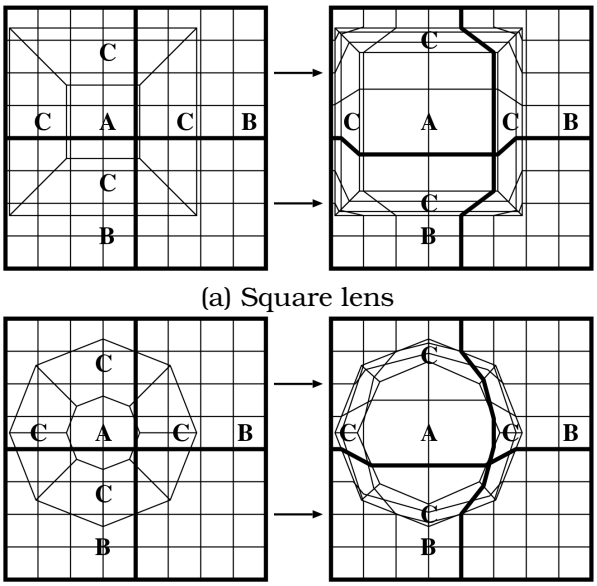
Image 4(c) uses the multiple-segment technique with eight segments. The compression is constant across the *warp* region. We have found, however, that the sharp edges at the *normal-warp* and *warp-zoom* borders can become quite distracting when the magnification lens is moved interactively. These sharp edges appear because the function across the regions is piece wise linear and, thus, only C^0-continuous.

Image 4(d) shows our solution to this problem: We generate the texture and geometric coordinates using a cubic Bezier curve, see [2], that is defined to interpolate value and gradient at the *normal* and *zoom* boundaries. This image is composed of 16 segments. The compression is no longer uniform: The transition from the border region to the fixed-magnification regions to either side is much smoother. This lens seems ideal for artifact-free zooming of an image. This technique requires more geometry than the linear technique (image 4(c)) which can become an issue for platforms where the CPU is the bottleneck.

4 Geometry

Figure 5 shows annotated square and circular geometry examples. Letters A-C denote separate transformation regions. Region A is the magnified center of the lens (*zoom*); the texture coordinates are simply modified as discussed in Section 3.1; region B shows non-magnified, outer region (*normal*); and region C shows the warp-magnified region that blends (*warp*) region (A) to region (B).

Image 5(a) shows the square geometry, and image 5(b) shows the circular geometry. For simplicity, this image shows only eight angular segments: all other images use 32 angular segments. The square geometry has the advantage



(a) Square lens

(b) Circular lens

Figure 5. Local geometric transformations: Letters A-C denote separate transformations: A: Magnified, center region of the lens; B: Non-magnified, outer region; C: Warp-magnified region that blends the inner, magnified, region (A) to the outer, non-magnified, region (B).

that there is much less geometry, both to generate and to clip. However, the clipping is only simple if the geometry is aligned with the base data, *e.g.*, a rectilinear data set. The significant disadvantage of this lens geometry is its unnatural appearance: the squareness, particularly at the corners, is a distracting artifact and can influence the interpretation of an image. The circular geometry allows us to approximate a circle (when tessellated to 32 angular segments). While this produces significantly more geometry, which must also be clipped, the overall visual quality seems much better and much more natural. One no longer sees the lens - one just sees the magnification.

4.1 Bounded Lens

Figure 6 shows that the circular tessellation for the lens does not need to be global: one can enclose the lens geometry inside a bounding square.

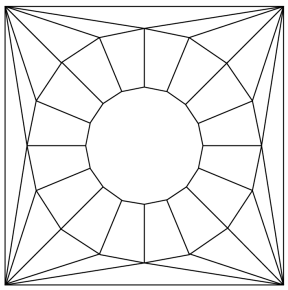


Figure 6. Circular geometry need not be global: a lens geometry can be "embedded" into a larger polygon (here, a square) or "stencil."

4.2 Clipping

If the lens or border is at the edge of an image tile, then the lens must be clipped in image/texture space, not the window. For the following discussion, we define the thicker lines in images 5(a) and 5(b) to be the tile boundaries of an image broken into 2x2 tiles. When using the square lens basis (Figure 5) that is aligned to the tile boundaries, clipping is very simple. However, when using a square lens basis that is not aligned with the tile boundaries, or when using the circular lens basis (image 5(b)), clipping becomes much more complicated and expensive. Examining the tile boundaries in image 5(b) shows that the tile boundaries that become concave. On the other hand, using the bounding-polygon technique discussed in the last section can significantly simplify the clipping operation by testing the bounding polygon for intersection with tile boundaries.

4.3 3D Geometry

Figure 7 shows how the geometry is generated for a volume. To render a volume, one renders a stack of textured polygons from back to front, where the viewing direction would ideally be perpendicular to the planes. For this image, we show the stack from the side to better illustrate its structure.

Adding a magnification lens is fairly simple. We use this terminology: the volume lens has *normal*, *warp*, and *zoom* regions. The *normal*, *warp*, and *zoom* regions in a plane are those regions of the plane that intersect the corresponding regions of the volume lens. The dashed lines

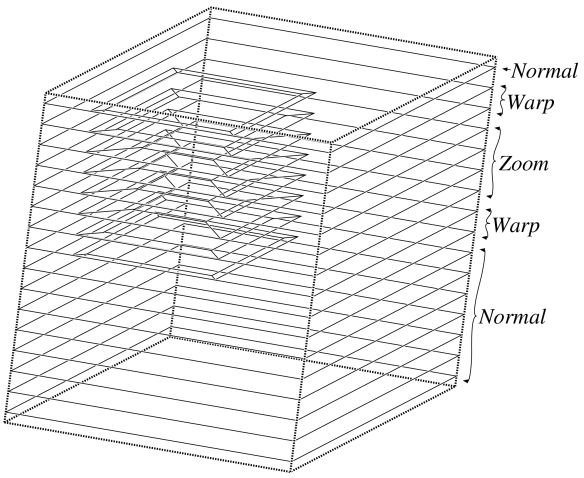


Figure 7. Stacking a square-lens geometry in 3D space.

in Figure 7 show the volume boundaries. The lens is located in the front-top-left corner of the volume. For simplicity, we show a square lens with 16 planes intersecting the volume. When planes do not intersect the *warp* or *zoom* regions of the lens (i.e., the *normal* regions), we use a single polygon. These are shown as solid green lines. When planes that intersect the *warp* region, we blend the volume's *warp* and *zoom* regions into the plane's *warp* and *zoom* regions. These are shown as solid blue lines. When planes that intersect the *zoom* region, we again blend the volume's *warp* and *zoom* regions into the plane's *warp* and *zoom* regions. These are shown as solid red lines.

Planes are not warped "out of the plane." This approach is simpler, and any warping in the view direction would be removed by orthographic projection. This aspect may be important for perspective projections, but we have not found this to be a an issue.

5 Results

5.1 Comparisons of Geometry and Texture Basis in 2D

Figure 8 shows examples generated with the square and circular lenses. This figure contrasts the simple, linear and cubic approaches. Table 2 summarizes the rendering performance, in frames per a second, for 2D imagery and textures. The window size is 500^2 pixels, with a total

	Images/Second			
Basis	Onyx/ IR	P3/ GeForce	Onyx/ IR	P3/ GeForce
Baseline	456	1067		
	Square		Circular	
Simple	425	1035	380	900
Linear	390	1012	357	908
Cubic	374	987	332	907

Table 2. Rendering performance in images-per-second for 2D imagery.

lens size of 250^2 pixels, the interior lens size is approximately 186^2 pixels, with a border size of 32 pixels. The square geometries have four angular segments, one for each face, and four linear or eight cubic radial (perpendicular to angular) segments. The circular method uses 32 angular segments, and four linear or eight cubic radial segments.

We used an SGI Onyx2 with four 195MHz MIPS R10K processors (only one used) and an Infinite Reality (IR) graphics subsystem; the second system that we used was a PC with an 866MHz Intel Pentium 3 processor with a GeForce2 GTS graphics card.

The texture was loaded into texture memory once, and then the textured geometry was rendered 100 times, and the average time computed. We have measured just the intra-frame rates, that is, the elapsed time for rendering all textured geometry.

The row titled "Baseline" refers to an unzoomed image, rendered with two triangles, to provide a maximum rendering rate. It is important to notice that the speeds are much closer to each other for the GeForce than for the IR - which is probably due to processor speed: The generation of the base geometry is strictly a function of the base processor, and the MIPS R10K/195MHz processor is much slower than the PIII/866MHz processor.

5.2 Comparisons of Geometry and Texture Basis in 3D

Figures 9 and 10 show the magnification-lens technique applied to two volumetric data sets. Figure 9 show the effect of square vs. circular basis and linear vs. cubic basis. Figure 10 shows the integration of the magnification lens with a

Figure 8. Zoom comparisons: Square/Circular and Simple/Linear/Cubic. Square-Linear is the simplest to process; Circular-Cubic produces the best quality; and Square-Cubic and Linear-Circular are shown for completeness.

	Skull
Baseline	5.9
Square, Simple	4.8
Square, Linear	4.6
Square, Cubic	4.4
Circular, Simple	4.3
Circular, Linear	3.9
Circular, Cubic	3.6

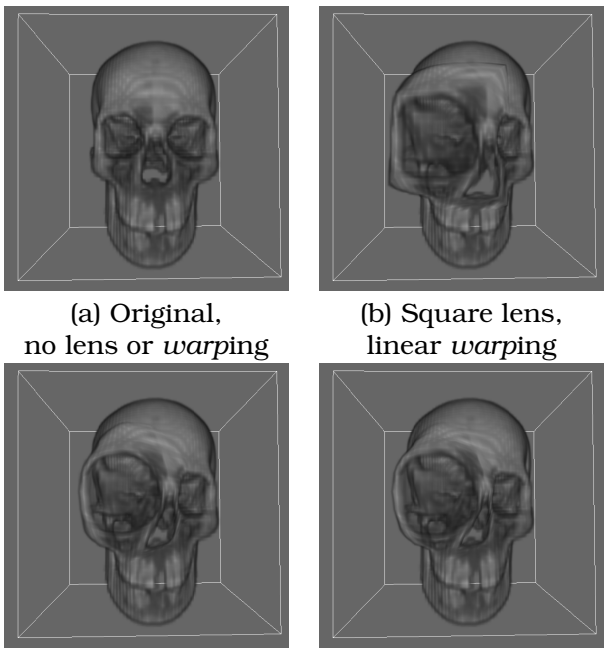
Table 3. Rendering performance in frames-per-second for skull data set (Figure 9).

clipping plane.

Table 3 summarizes the performance for the skull data set, shown in Figure 9. For the lenses used in Figure 9, the linear basis uses four radial segments while cubic basis uses eight radial segments; the circular basis use 16 angular segments. The texture was loaded to texture memory once, and then the textured geometry rendered several times, and the average time computed. We have measured just the intra-frame rates, that is, the elapsed time for rendering all textured geometry. These time are measured on an Onyx2 with InfiniteReality graphics with one Raster Manager. This test was not done on the GeForce, as our algorithm uses only 3D textures, and the GeForce2 does not support 3D textures.

6 Conclusions and Future Work

We have presented an algorithm for performing a magnification-lens technique for volume visualization. We use homogeneous texture coordinates and special geometries to implement a magnifying glass to provide a user with the ability to zoom in on small regions of a very large data set, while providing a smooth transition to un-zoomed regions. This technique is quite



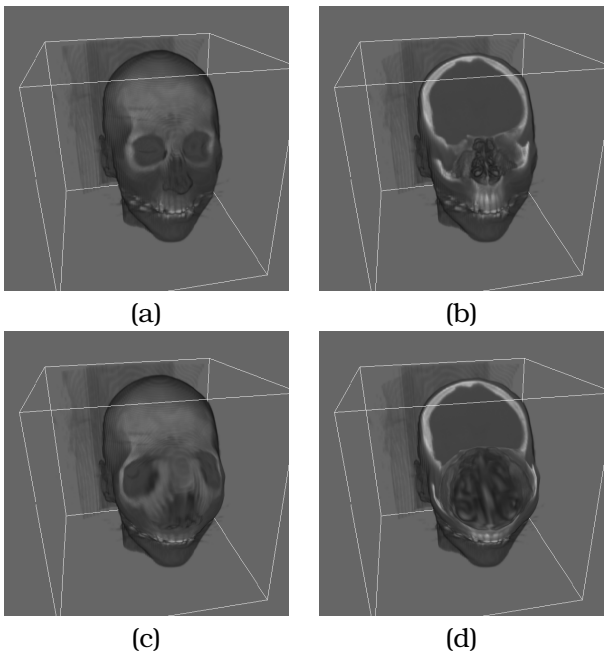
(a) Original, no lens or *warp*ing

(b) Square lens, linear *warp*ing

(c) Circular lens, linear *warp*ing

(d) Circular lens, cubic *warp*ing

Figure 9. Skull data set rendered with a lens located in the right orbit.



(a)

(b)

(c)

(d)

Figure 10. Clipped CT data set. Image (a) shows the original, undistorted head; (b) shows the head with clipping plane exposing the nasal passages; (c) shows the head with a circular-cubic lens; and (d) shows the head with clipping plane and magnification, exposing and enlarging the nasal passages.

effective for zooming into large data sets, provided that the system used for rendering is not generally compute-bound (in contrast to fill-rate bound). When a clipping plane is added, we have a very powerful technique for examining small details of a very large data set.

We plan to apply this method to our multiresolution volume visualization system [8, 7, 6]. However, a significant hurdle is the efficient clipping of the lens against individual tiles. If the thick lines in Figures 5(a) and 5(b) delimit tile (individual texture) boundaries, hardware-based clipping becomes very difficult.

A software technique that clips based on texture coordinates might work, but then efficiency becomes a concern. Second, new graphics cards (*e.g.*, Nvidia's GeForce3) allow dependent texture look-ups: One could implement the schemes discussed here by defining an intermediate texture that performs the warping. This would vastly reduce the amount of geometry required. Third, blending between two images, or data sets, through the *warp* region will be necessary because the *normal* and *zoom* regions typically use different images, one being a higher resolution version of the other. We also would have to extend this method to *warp* hierarchies that can blend between several levels of resolution.

Acknowledgements

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors ex-

pressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

This work was supported by the National Science Foundation under contracts ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis, and the members of the Data Analysis and Exploration thrust of the Center for Applied Scientific Computing (CASC) at Lawrence Livermore National Laboratory.

References

[1] Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony DeRose. Toolglass and Magic Lenses: The See-through Interface. In *SIGGRAPH '93*, volume 27, pages 73–80, August 1993.

[2] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. fourth edition, Academic Press, Boston, 1997.

[3] T. Alan Keahey. The Generalized Detail-In-Context Problem. In *IEEE Information Visualization 1998*, pages 44–51. IEEE, 1998.

[4] T. Alan Keahey. Visualization of High-dimensional clusters using Nonlinear Magnification. In *Visual Data Exploration and Analysis VI - Proceedings of the SPIE*, pages 228–235, 27–28 January 1999.

[5] Yair Kurzion and Roni Yagel. Interactive Space Deformation with Hardware-Assisted Rendering. *IEEE CG&A*, 17(5):66–77, September/October 1997.

[6] Eric C. LaMar, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Texturing-based Rendering of Arbitrarily Oriented Cutting-Planes. In *Data Visualization 2000*, pages 105–114. EUROGRAPHICS/IEEE, 29–30 May 2000.

[7] Eric C. LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Hardware Texturing-based Volume Visualization. In *Visual Data Exploration and Analysis*, pages 365–374. SPIE, January 2000.

[8] Eric C. LaMar, Kenneth I. Joy, and Bernd Hamann. Multi-Resolution techniques for Interactive Hardware Texturing-based Volume Visualization. In *IEEE Visualization '99*, pages 355–361, 25-29 October 1999.

[9] Uwe Rauschenbach and Heidrun Schumann. Demand-driven image transmission with levels of detail and regions of interest. *Computers and Graphics*, 23(6):857–866, December 1999.

[10] Manojit Sarkar and Marc H. Brown. Graphical Fisheye Views of Graphs. Technical Report 84, Digital Equipment Corporation, Systems Research Centre, 17 March 1992.

[11] Manojit Sarkar, S. Snibbe, Oren J. Tversky, and Steven P. Reiss. Stretching the Rubber Sheet: A Metaphor for Viewing Large Layouts on Small Screens. Technical Report CS-93-39, Dept. of CS, Brown University, September 1993.

[12] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *ACM Siggraph '92*, volume 26, pages 249–252, 26–31 July 1992.

[13] Colin Ware and Marlon Lewis. The DragMag Image Magnifier. In *ACM CHI '95*, volume 2, pages 407–408, 1995.