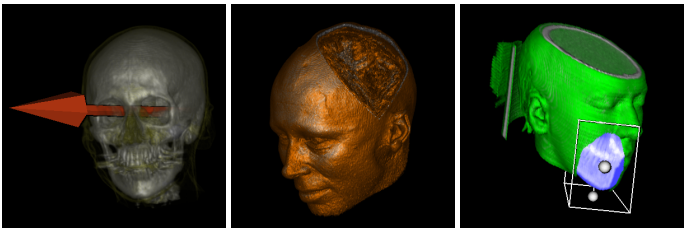
The Inverse Warp: Non-Invasive Integration of Shear-Warp Volume Rendering into Polygon Rendering Pipelines

Stefan Bruckner* Dieter Schmalstieg† Helwig Hauser‡
M. Eduard Gröller*

*Institute of Computer Graphics and Algorithms
†Institute for Software Technology and Interactive Systems
Vienna University of Technology, Austria

‡VRVis Research Center
Vienna, Austria



Abstract

In this paper, a simple and efficient solution for combining shear-warp volume rendering and the hardware graphics pipeline is presented. The approach applies an inverse warp transformation to the Z-Buffer, containing the rendered geometry. This information is used for combining geometry and volume data during compositing. We present applications of this concept which include hybrid volume rendering, i.e., concurrent rendering of polygonal objects and volume data, and volume clipping on convex clipping regions. Furthermore, it can be used to efficiently define regions with different rendering modes and transfer functions for focus+context volume rendering. Empirical results show that the approach has very low impact on performance.

1 Introduction

Shear-Warp factorization [6] is generally considered to be one of the most efficient methods for software-based volume rendering [10]. It has proven to achieve interactive frame-rates comparable to methods that exploit hardware acceleration, but still maintains the flexibility of a software solution. For applications which cannot take advantage of the features provided by the latest graphics hardware, it therefore provides a reasonable alternative. Images created with this algorithm are usually rendered as billboard textures with graphics APIs such as OpenGL [18].

However, a problem arises when it is desired to integrate shear-warp volume rendering with conventional geometry rendering. When polygons intersect the volume, the ”flat” nature of the texture becomes visible, which is disturbing and partly destroys the three-dimensional impression. Moreover, augmenting the volume with text, markers, etc. as it is often useful in medical applications, is limited.

In volume visualization, compositing is commonly used to model emission and absorption effects [9]. This discrete approximation accumulates color contributions (emission) which are weighted by translucency (absorption). Compositing allows to simultaneously visualize surfaces and interior structures. For combining geometry and volume rendering, emission and absorption effects of the geometry have to be considered during this process.

In this paper we describe a method to produce renderings of intersected volume data and opaque geometry through a simple modification of the

*{bruckner|meister}@cg.tuwien.ac.at.at
†schmalstieg@ims.tuwien.ac.at
‡Hauser@VRVis.at

shear-warp algorithm. We also show that our approach can be used to perform volume clipping. Additionally, a focus+context volume rendering approach can be realized using this method.

In Section 2, we describe other approaches that have been presented for concurrent rendering of geometry and volume data. Section 3 describes the inverse warp transformation, which is the basis for the applications discussed in Section 4. In Section 5, we present the results and discuss the performance of our method. Finally, this paper is concluded in Section 6.

2 Related Work

Much work has been done on rendering geometry and volume data concurrently. The main problem that has to be considered is the difference in representation. While volume data is a set of samples, geometry data is analytically defined. Volume data is usually represented on a three-dimensional grid. Geometry data is a set of analytically defined surfaces. In real-time rendering these are polygonal meshes.

One idea is to convert polygon and volume data into a common representation. Algorithms such as Marching Cubes [8], extract surfaces from volume data. These surfaces can then be rendered together with the geometry, using the graphics hardware's Z-Buffer to ensure correct visibility. However, since only surfaces are extracted, a lot of information is lost during this process. The process of converting geometry into volume data is referred to as voxelization [15]. This approach does not have the problem that information about the interior of objects is lost. However, since applications use a huge number of polygons, it can be very time-consuming when done in software.

The drawbacks of these approaches have lead to algorithms that simultaneously operate on geometry data and volume data [7, 4]. They combine rasterization of the geometry and volume rendering to produce a hybrid rendering.

Lacroute and Levoy [6] introduced shear-warp factorization, an efficient volume rendering algorithm. In his thesis [5], Lacroute suggested a hybrid algorithm that simultaneously performs compositing and rasterization for each scanline. However, this software approach cannot compete with the rasterization capabilities of modern graphics hardware, available in every standard PC. Furthermore, it cannot be integrated with standard graphics APIs.

In the work by Zakaria and Zaman [19] and Schmidt et al. [13] the geometry is rasterized into *sheared-object space* (this means that the shear transformation of the shear-warp algorithm is applied to the geometry). The color and Z-Buffer established during rasterization are then used in compositing to account for geometry contributions to the rendering. Finally, the two-dimensional warp is applied to the *intermediate image*. The rasterization can be performed using graphics hardware, by rendering the intersecting geometry into a non-visible buffer and reading out the color and Z-Buffer.

However, existing systems need considerable modifications for incorporating this method: Intersecting objects have to be determined and excluded from the normal rendering process. For a large number of objects, this might require to perform intersection tests for all objects or the introduction of advanced spatial data structures.

Our method does not suffer from this drawback, making it more applicable for integration into existing systems. All geometry is rendered first. Then the region of the hardware's Z-Buffer corresponding to the projection area of the volume is read out. An inverse warp is applied to this partial Z-Buffer, to transform it into *sheared-object space*. As suggested by Schulze et al [14], this inversely warped Z-Buffer is used to determine intersecting geometry.

3 Method Overview

In this section, we describe the basics of our method for hybrid shear-warp volume rendering. First, we introduce the terminology of the shear-warp factorization. We then describe the inverse warp transformation of the Z-Buffer, which allows to integrate geometry into the volume rendering algorithm.

3.1 Shear-Warp Factorization

The basic idea of shear-warp factorization [5, 6] is that the view matrix is factorized into a shear and a warp matrix. Also a permutation according to the principal viewing axis is involved, but for simplicity we will disregard it here.

Applying the shear transformation to the volume means transforming each volume slice in a way,

so that all viewing rays are parallel to the principal viewing axis. The coordinate system defined by this property is called *sheared-object space*. For parallel projections, this means a translation of every volume slice. For perspective projections, each slice has to be scaled as well. Since all viewing rays are parallel to the principal viewing axis in *sheared-object space*, the algorithm can process the volume in a slice-by-slice, scanline-by-scanline manner, compositing into a so-called *intermediate image*. This allows cache-efficient access to the volume data and is the basis for several high-level optimizations [5, 6, 11].

After compositing has taken place, a warp is performed which transforms the *intermediate image* to the final image. However, since the *intermediate image* contains color information, only the two-dimensional part of this transformation has to be considered.

3.2 Inverse Warp Transformation

Today's graphics hardware uses the Z-Buffer algorithm [16] for displaying polygons with correct visibility. In general, for each fragment (a rasterized portion of a polygon, attributed with location, color, etc.), the fragment is only written into the frame buffer if its distance to the image plane is lower than the distance stored in the Z-Buffer. If this is the case, the fragment's depth value is written into the Z-Buffer, overwriting the old value at that location. Graphics APIs, such as OpenGL [18], allow to read out the graphics card's Z-Buffer into main memory.

The Z-Buffer established when rendering geometry can be used within the volume rendering algorithm to perform several operations, such as hybrid volume rendering or clipping (see Section 4). For using the information provided by the Z-Buffer during the compositing phase of the shear-warp algorithm, it has to be transformed into *sheared-object space*.

One way to perform this operation would be to adjust the transformation matrix such that rasterization of the geometry is performed directly into *sheared-object space*. However, this approach has several drawbacks: Rasterization of polygons intersecting the volume has to be performed twice, one time, using the original transformation in order to establish the actual frame buffer, and once for transforming them into *sheared-object space*. Therefore, for scenes containing a large number of polygons, the portion of polygons that actually intersect the volume has to be identified - otherwise the whole scene would have to be rendered twice. This can be accomplished by using the spacial data structures such an application surely would incorporate (e.g. bounding volume hierarchies). However, it complicates the task of integrating the algorithm into existing systems.

```
// Input:     M           (transformation matrix)
//            source      (source image)
// Output:    destination (destination image)

for y = [0 .. destination.height-1]
{
  for x = [0 .. destination.width-1]
  {
    t = M^(-1) * (x,y)
    destination[x][y] = source[t.x][t.y]
  }
}
```

Listing 1: Two-dimensional backward-mapped warp

Therefore, we chose an approach which relies on no information about the scene apart from the Z-Buffer that has been established during rendering.

Our method uses the inverse warp transformation to transform the Z-Buffer from *image space* to *sheared-object space*. However, in contrast to the warp of the *intermediate image* from *sheared-object space* to *image space*, a two-dimensional transformation is not sufficient. One can interpret the Z-Buffer as a set of point samples of the scene geometry. The x and y coordinates of a sample point are implicitly defined by its location in the buffer, whilst the z coordinate is stored in the buffer itself. Performing just a two-dimensional warp would only cause a two-dimensional translation of each point, rather than correctly transforming it. In the remainder, we will refer to the actual Z-Buffer as *source* and to the result of the inverse warp as *destination*, treating both as images containing depth values. This highlights the difference between a two-dimensional and a three-dimensional warp and suggests an implementation.

Common methods for image warping use backward-mapping. In backward-mapped warping, the destination pixels are inversely mapped to the source image and sampled accordingly, as depicted in Listing 1. However, since we need the depth value at each sample location in order to compute

```
// Input:     M           (transformation matrix)
//            source      (source image)
// Output:    destination (destination image)

for y = [0 .. source.height-1]
{
  for x = [0 .. source.width-1]
  {
    t = M * (x,y,source[x][y])
    destination[t.x][t.y] = t.z
  }
}
```

Listing 2: Three-dimensional forward-mapped warp

the transformation, backward-mapping cannot be used. Instead, we use a forward-mapping algorithm [1] as depicted in Listing 2.

This causes some problems. The inverse warp is performed on a discrete set of samples. Thus, proper reconstruction has to be done. We use a rectangular footprint, scaled according to the ratio of the dimensions of the volume's projection on the image plane and the dimensions of the *intermediate image*. For perspective projection, the footprint is scaled according to the depth value as well. This ensures that no holes occur in the transformed image. Though this is a rather coarse approximation, it has proven to provide sufficient quality.

Yet another problem remains: Several *source* locations can map to the same *destination* location. When not handled correctly, this introduces very disturbing visibility errors. We therefore use a minimum operator to combine *destination* values and *source* values (the *destination* is initialized with infinity). This corresponds to the Z-Buffer algorithm - the value closest to the viewer is chosen.

The result of the inverse warp is a depth map in *sheared-object space* coordinates of the same size as the *intermediate image*. Therefore, finding the depth value for a sample location can be accomplished by looking up the depth map's value using the current indices for the *intermediate image*. Negative values in this map correspond to geometry in front of the volume, values that exceed the volume's dimension along the principal viewing axis, correspond to geometry behind the volume. All other values correspond to geometry that intersects the volume. Figure 1 illustrates the use of the inverse warp transformation for combining volume and geometry rendering.

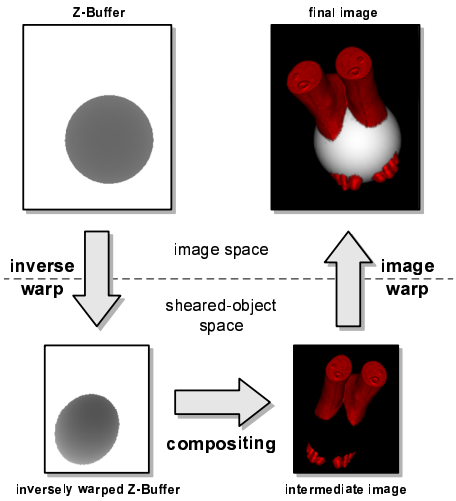


Figure 1: Integration of geometry and volume rendering using the inverse warp transformation

4 Applications

4.1 Hybrid Volume Rendering

Concurrent display of intersecting volume data and polygons is often desired in virtual environments. It allows to display augmentations, markers or labels located next to structures of special interest within the volume, enabling the user to perceive this information in the correct context. Virtual objects aligned with real-world props can be used to provide a three-dimensional interface [3]. Using the

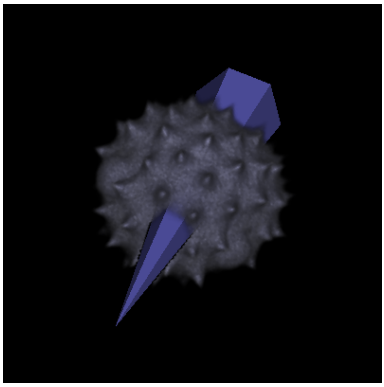


Figure 2: An example of hybrid volume rendering

```
// Input:     volume      (volume data)
//            depthmap    (inv. warped Z-Buffer)
// Output:    final       (final image)

for k = [0 .. volume.dimensions.z - 1]
{
  for j = [0 .. volume.dimensions.y - 1]
  {
     for i = [0 .. volume.dimensions.x - 1]
     {
       [u,v] = Shear(i,j,k)
       [i,j,u,v] = Skip(i,j,u,v)

       if (k >= depthmap[u][v])
         Terminate(intermediate[u][v])
       else
       {
         Composite(intermediate[u][v],
                   volume.data[i][j][k])

         if (intermediate[u][v].opacity >= 1.0)
           Terminate(intermediate[u][v])
       }
     }
  }
}

Warp(final,intermediate);
```

Listing 3: Hybrid shear-warp algorithm

inverse warp transformation, hybrid volume rendering can be easily implemented.

All geometry is rendered first, then the Z-Buffer is read out and is inversely warped. During the volume rendering algorithm, rays intersecting geometry are terminated. A test has to be performed at every sampling location, comparing the current slice index with the inversely warped Z-Buffer. If the value is equal or higher than the current slice index, no more compositing has to be performed for the corresponding *intermediate image* pixel - the ray can be terminated. When runlength-encoding of the *intermediate image* is used, as proposed by Lacroute [5], runs of voxels only contributing to already terminated rays can be efficiently skipped.

Listing 3 gives the pseudo-code for the hybrid shear-warp algorithm. Note that the only change that is required (depicted in **boldface**), is a simple lookup in the inversely warped Z-Buffer. This technique can be applied to any existing implementation of the shear-warp algorithm.

The advantage of this approach is that no information about the geometry, apart from the Z-Buffer, is needed within the volume rendering algorithm. This allows hybrid volume rendering to be integrated very easily into existing geometry-based systems, such as virtual or augmented reality environments. The only constraint is, that the volume has to

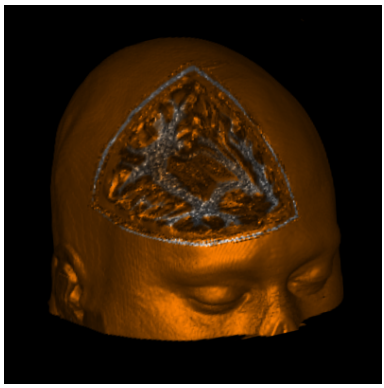


Figure 3: An example of volume clipping

be rendered after the geometry, since the algorithm needs the information stored in the Z-Buffer.

We have successfully integrated our hybrid approach into the Visualization Toolkit (VTK), a data-flow based visualization library and Studierstube [12], a collaborative augmented reality environment based on Open Inventor. An example image of hybrid shear-warp volume rendering can be seen in Figure 2. It displays a volume rendered daisy pollen granule (acquired by laser scanning confocal microscopy), intersected by a polygonal cone.

4.2 Volume Clipping

As presented by Weiskopf et al [17], volume clipping for convex clipping regions can be implemented by testing against the Z-Buffers established by rendering both, front faces and back faces separately. Sampling locations that lie within the clipping region are simply skipped.

This concept can be used in conjunction with the inverse warp transformation, to introduce such clipping regions to the shear-warp algorithm (see Figure 3, which displays a MRI dataset clipped with a polygonal box). Listing 4 depicts the innermost loop of a shear-warp algorithm with support for clipping regions.

4.3 Focus+Context Volume Rendering

Extending the previous approach, the concept of focus+context well known in information visualization can be applied to volume visualization. Rather

```
[u,v] = Shear(i,j,k)
[i,j,u,v] = Skip(i,j,u,v)

if (k < depthmap_front[u][v] ||
    k > depthmap_back[u][v])
{
  Composite(intermediate[u][v],
            volume.data[i][j][k])

  if (intermediate[u][v].opacity >= 1.0)
    Terminate(intermediate[u][v])
}
```

Listing 4: Innermost loop of a shear-warp algorithm supporting clipping regions

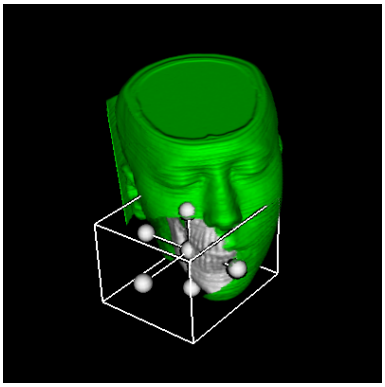


Figure 4: An example of focus+context volume rendering

than just clipping the geometrically defined region, the render mode is adjusted inside the region. Hauser et al introduced two-level volume rendering [2], a way to combine different render modes within a single dataset.

Using geometry to define regions of interest further extends the use of this idea. Instead of assigning different render modes to presegmented objects within the dataset, geometric regions of interest can be used to provide tools, such as magic lenses, that can be interactively repositioned and deformed. An example can be seen in Figure 4, where an interactive tool allows to render parts of the data set with different transfer functions. Listing 5 depicts the innermost loop of a shear-warp algorithm that supports focus+context volume rendering.

```
[u,v] = Shear(i,j,k)
[i,j,u,v] = Skip(i,j,u,v)

if (k < depthmap_front[u][v] ||
    k > depthmap_back[u][v])
  CompositeNormal(intermediate[u][v],
                  volume.data[i][j][k])
else
  CompositeClipped(intermediate[u][v],
                   volume.data[i][j][k])

if (intermediate[u][v].opacity >= 1.0)
  Terminate(intermediate[u][v])
```

Listing 5: Innermost loop of a focus+context shear-warp algorithm

5 Results and Discussion

The advantage of our approach is that it is independent of both, volume size and geometrical complexity. The performance of the inverse warp is proportional to the size of the volume's projection on the image plane.

One potential problem of our approach is that it requires to read the Z-Buffer back into main memory. While modern graphics hardware has tremendous rasterization and pixel fill capabilities, transferring data from the graphics card's memory to main memory can be a costly operation. Especially on low-cost graphics hardware, the so-called "2D-path" is not optimized. We have therefore tested the performance of the Z-Buffer read on various common graphics boards. Table 1 shows the results of our measurements for Z-Buffer read and inverse warp. We have included timings for three common graphics chipsets to display the variations in Z-Buffer read performance. Of those only the ProSavage DDR, a mobile solution with shared-memory architecture, shows substantial problems. Even the GeForce 4 MX 440, a low-cost consumer product, provides sufficient performance.

During the volume rendering algorithm itself only simple testing operations have to be performed. To improve cache coherency, the inversely warped z value for each pixel can be stored as an element of the *intermediate image*, instead of storing it in a separate buffer. Using this approach the test required for hybrid volume rendering introduced no detectable performance impact in our implementation.

For evaluating image quality, we have compared renderings of polygonal objects with voxelized versions of these objects, both intersected by geometry.

Figure 5 shows images of a highly tesselated sphere intersected by a cone and images of the same scene using a voxelized version of the sphere. Differences at the intersecting regions are visible in the magnification. The visible artifacts are caused by the reconstruction performed during the inverse warp transformation (as described in Section 3.2). These artifacts could be reduced by using other reconstruction techniques, however, this would also affect performance. Since the appearance is also improved by texture blending, which causes smoothing, we consider our choice to be a good trade-off between quality and speed.

One limitation is that our method can only be used for opaque geometry. It is not possible to restore original alpha values from the frame buffer, after blending has been performed. While it is possible to introduce a limited number of transparency levels, e.g. by rendering translucent objects separately, easy integration, one of the main features of our approach, severely suffers from this. It has been suggested that a possible solution to this problem is a Zlist-Buffer, which stores z and alpha values for all surfaces seen through an image pixel [19]. If such a buffer was to be implemented in hardware, our approach could be extended to support translucent geometry in a straight-forward manner.

6 Conclusion

The method we have presented allows to combine polygons and shear-warp volume rendering by applying an inverse warp transformation to the Z-Buffer. We have presented three applications of this technique, hybrid volume rendering, volume clipping and focus+context volume rendering. Our method is independent of volume size and geometric complexity.

The key feature of our approach is that it allows to integrate volume rendering into existing geometry-based systems, since no information about the geometry apart from the Z-Buffer is needed. This advantage enabled us to easily integrate our algorithm into Studierstube, a distributed augmented reality environment.

Future work will include the development of more advanced interaction techniques and interactive volume visualization tools based on the approach presented in this paper.

7 Acknowledgments

This work was partially sponsored by the Austrian Science Fund under contract no. Y193 and contract no. P15217. Parts of this work have been done in the scope of VRVis basic research on visualization (http://www.VRVis.at/vis/) which is funded by an Austrian research program called K plus.

We thank Thomas Theußl for his help in preparing this document. We also thank Tamer Fahmy and Rudolf Seeman for their support in setting up Studierstube.

References

[1] B. Chen, F. Dachille, and A. Kaufman. Forward image mapping. In *Proceedings of IEEE Visualization 1999*, pages 89–96, 1999.

[2] H. Hauser, L. Mroz, G. Bischi, and M. Eduard Gröller. Two-level volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 7(3):242–252, July 2001.

[3] K. Hinckley, R. Pausch, J. Goble, and N. Kassell. Passive real-world interface props for neurosurgical visualization. In *Proceedings of ACM CHI 1994 Conference on Human Factors in Computing Systems*, volume 1 of *Interacting in 3-D*, pages 452–458, 1994.

[4] K. Kreeger and A. Kaufman. Mixing translucent polygons with volumes. In *Proceedings of IEEE Visualization 1999*, pages 191–198, 1999.

[5] P. Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. PhD thesis, Stanford University, 1995.

[6] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, 1994.

[7] M. Levoy. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications*, 10(2):33–40, March 1990.

[8] W. Lorensen and H. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169, 1987.

[9] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, June 1995.

[10] M. Meißner, J. Huang, D. Bartz, K. Mueller, and R. Crawfis. A practical evaluation of popular volume rendering algorithms. In *Proceedings of the IEEE Symposium on Volume Visualization 2000*, pages 81–90, 2000.

[11] L. Mroz and H. Hauser. RTVR - a flexible java library for interactive volume rendering. In *Proceedings of IEEE Visualization 2001*, pages 279–286, 2001.

Final image size	Z-Buffer read			Inverse warp
	ProSavage DDR	GeForce 4 MX 440	GeForce 4 Ti 4600	Pentium 4 2.4 GHz
128 x 128	0.01112	0.00109	0.00063	0.00219
256 x 256	0.04477	0.00391	0.00203	0.00875
384 x 384	0.09615	0.00731	0.00472	0.01812
512 x 512	0.18366	0.01713	0.00797	0.03656
640 x 640	0.28632	0.02693	0.01249	0.05625
768 x 768	0.40148	0.03855	0.01781	0.08187

Table 1: Performance evaluation. The table lists the times (in seconds) for reading the Z-Buffer and performing the inverse warp for several final image sizes. All times are average values over 100 iterations.

[12] D. Schmalstieg, A. Fuhrmann, G. Hesina, Zs. Szalavári, L. M. Encarnação, M. Gervautz, and W. Purgathofer. The Studierstube augmented reality project. *PRESENCE - Teleoperators and Virtual Environments*, 11(1):32–54, February 2002.

[13] A. Schmidt, M. Gattass, and P. Carvalho. Combined 3D visualization of volume data and polygonal models using a shear–warp algorithm. *Computers and Graphics*, 24(4):583–601, August 2000.

[14] J. Schulze, R. Niemeier, and U. Lang. The perspective shear-warp algorithm in a virtual environment. In *Proceedings of IEEE Visualization 2001*, pages 207–214, 2001.

[15] M. Sramek and A. Kaufman. Alias-free voxelization of geometric objects. *IEEE Transactions on Visualization and Computer Graphics*, 5(3):251–267, July/ September 1999.

[16] T. Theoharis, G. Papaioannou, and E. Karabassi. The magic of the Z-buffer: A survey. In *Proceedings of WSCG 2001*, pages 379–386, 2001.

[17] D. Weiskopf, K. Engel, and T. Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. In *Proceedings of IEEE Visualization 2002*, pages 93–100, 2002.

[18] M. Woo, J. Neider, T. Davis, and OpenGL Architecture Review Board. *OpenGL programming guide: the official guide to learning OpenGL, version 1.1*. Addison-Wesley, Reading, MA, USA, second edition, 1997.

[19] M. Zakaria and M. Saman. Hybrid shear-warp rendering. In *Proceedings of International Conference on Visual Computing 1999*, pages 144–153, 1999.

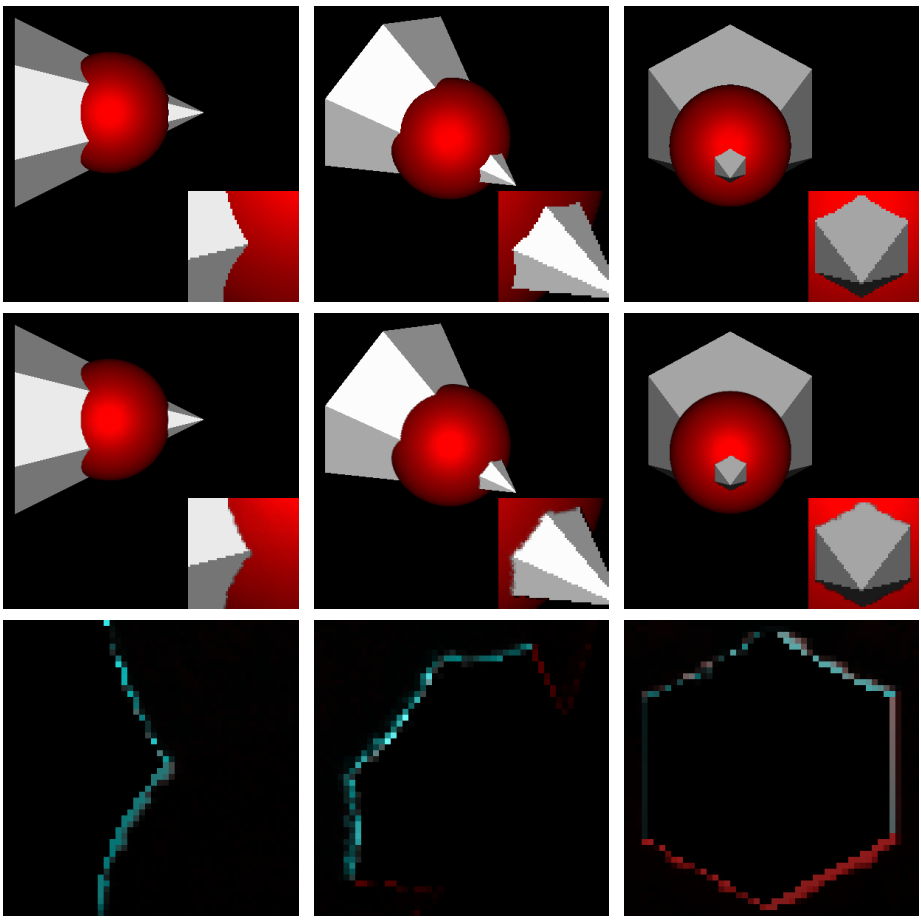


Figure 5: Comparison of image quality. **Top:** Polygonal sphere intersected by polygonal cone, rendered using OpenGL. **Middle:** Voxelized sphere intersected by polygonal cone, rendered using our hybrid shear-warp algorithm (sharp transfer function). **Bottom:** Difference images of the magnified regions.