# The Perspective Shear-Warp Algorithm In A Virtual Environment

Jürgen P. Schulze* Roland Niemeier† Ulrich Lang*

*High Performance Computing Center Stuttgart (HLRS)

†science + computing ag

## Abstract

Since the original paper of Lacroute and Levoy [9], where the shear-warp factorization was also shown for perspective projections, a lot of work has been carried out using the shear-warp factorization with parallel projections. However, none of it has proved or improved the algorithm for the perspective projection. Also in Lacroute's Volpack library, the perspective shear-warp volume rendering algorithm is missing.

This paper reports on an implementation of the perspective shear-warp algorithm, which includes enhancements for its application in immersive virtual environments. Furthermore, a mathematical proof for the correctness of the permutation of projection and warp is provided, so far a basic assumption of the shear-warp perspective projection.

**CR Categories:** I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

**Keywords:** Volume Rendering, Perspective Shear-Warp, Virtual Environments

## 1 INTRODUCTION

The algorithms based on the shear-warp factorization, especially the one developed by Lacroute [9], are among the fastest volume rendering algorithms. They have often been compared to hardware accelerated volume rendering techniques, such as general purpose graphics boards with texturing acceleration [1], or specialized volume rendering hardware [6, 11, 10].

Although on single processor machines the shear-warp algorithm is usually slower than hardware supported solutions, the shear-warp's good scalability allows it to be competitive on multiprocessor machines [8]. So far, volume rendering in virtual environments (VE) with multiple screens is dominated by texturing hardware approaches. Due to the necessary high resolution of about $1000^2$ pixels per stereo screen, the image quality is limited by the pixel fill rate. Thus, it degrades considerably when the volume object fills a significant part of the screens, because subsampling algorithms need to be applied to maintain an interactive frame rate. In contrast, shear-warp rendering speed does not depend on the output image size. At comparable output image quality in a $1000^2$ window, our PC system (see section 4) renders a $64^3$ data set at 2.2 frames per second (fps) using the shear-warp algorithm, compared to 1.3 fps with texture hardware acceleration.

Many extensions, like stereo rendering [5], parallel algorithms [8], clipping planes [16], and performance improvements [4] have been added to the shear-warp algorithm for parallel projections. However, only a few implementations or enhancements of the shear-warp algorithm for perspective projections were reported, e.g., an improvement of the warp [2], and none of them address the compositing.

Perspective volume rendering has well known advantages. Depth information perceived by perspective projection is important, e.g., for radiation therapy planning. Also for immersive virtual environments a restriction to parallel projection algorithms generates depth ambiguities. For CAVE-like environments [3], or for any other non-flat multi-projection environments, perspective projection is a requirement.

The shear-warp algorithm processes volume data arranged on cartesian grids. It is based on the idea of factorizing the viewing matrix into a shear and a 2D warp component, with the projection done in between. After applying the shear matrix, the volume slices are projected and composited to a 2D sheared image. The shear step enables the algorithm to operate in object space with high memory locality, which optimizes the usage of RAM caching mechanisms. The warp being performed in 2D space by generating the final image from the intermediate image, decreases the computational complexity considerably, compared to a 3D operation.

Lacroute's shear-warp thesis [7] adds some ideas to further increase rendering speed. Both the volume data and the intermediate image data are run-length encoded (RLE) to minimize the number of memory accesses, to save storage space, and to further increase memory locality. The RLE encoded volume data are stored in memory three times, once for each principal coordinate axis. Shading is performed by precomputing a normal vector for each volume element (voxel) and assigning colors using a look-up table. A fast classification can be done by using an octree based algorithm instead of RLE encoding.

This paper introduces the first application of the perspective shear-warp algorithm in a VE. The mathematical foundation of the factorization is proved and an implementation is presented. The implementation features some developments which were necessary for the algorithm to be used in a VE, such as a clipping plane, adjustable rendering quality, usage of texturing hardware for the warp, and limited concurrent display of polygons. The implementation was integrated into the VIRVO system [13], which allows its direct comparison to texture based algorithms. VIRVO provides both a system independent Java GUI for work on the desktop and a plugin for COVER, which is the virtual reality rendering subsystem of the visualization software COVISE [12]. The virtual environment development was done in the CUBE, the University of Stuttgart's 4-sided CAVE-like device, located at the High Performance Com-

*Allmandring 30, 70550 Stuttgart, Germany
Email: {schulze|lang}@hlrs.de
†Hagellocher Weg 73, 72070 Tübingen, Germany
Email: roland@science-computing.de

Presented at IEEE Visualization 2001
October 21 - October 26, 2001
San Diego Paradise Point Resort, San Diego

puting Center (HLRS).

In section 2, we describe the mathematical background of the shear-warp algorithm. Section 3 addresses specific implementation requirements for the algorithm when used in VEs. Section 4 provides performance numbers and a comparison of the shear-warp algorithm to texture hardware based techniques.

## 2 THE PERSPECTIVE SHEAR-WARP

In this section, the factorization for the perspective viewing transformation is reviewed in brief. It basically follows Lacroute's derivation [7]. Furthermore, the permutation of projection and warp is proved. Finally, warp performance of the parallel and the perspective algorithm is compared.

### 2.1 Conventions

Lacroute uses four different coordinate systems in his derivation. We think that for an easier understanding of the algorithm, six coordinate systems are to be distinguished. These are listed in table 1, which also assigns a unique single character identifier to each of them. Additionally, the coordinate systems are illustrated in figure 1.

Table 1: Coordinate systems.

| | | |
|---|---|---|
| o | object space | actual coordinate system of the volume data set |
| s | standard object space | coordinate system after permutation of object space coordinate axes |
| d | deformed space | 3D coordinates after shear |
| i | intermediate image space | 2D coordinates within intermediate image |
| w | world coordinate space | 3D world coordinates |
| v | viewport space | 2D output window coordinates |

In the following, transition matrices between coordinate systems carry the names of the corresponding source and destination coordinate systems, e.g., the transition from coordinate system $o$ to $w$ would be named $M^{ow}$. The inverse matrix $(M^{ow})^{-1}$ would be $M^{wo}$. Vector elements are named $x, y, z, w$.

### 2.2 Prerequisites

The goal of the factorization is to obtain the shear matrix $M^{oi}$ and the warp matrix $M^{iv}$ so that the viewing matrix is:

$$M^{ov} = M^{iv} * M^{oi}$$

The camera parameters define the projection from world coordinates to viewport space $M^{wv}$, so the transformation from object space to world coordinates is:

$$M^{ow} = M^{vw} * M^{ov}$$

### 2.3 Factorization

This section briefly explains the required computation steps for the factorization of the perspective viewing matrix.

First of all, the object space eye position $e^o$ has to be found:

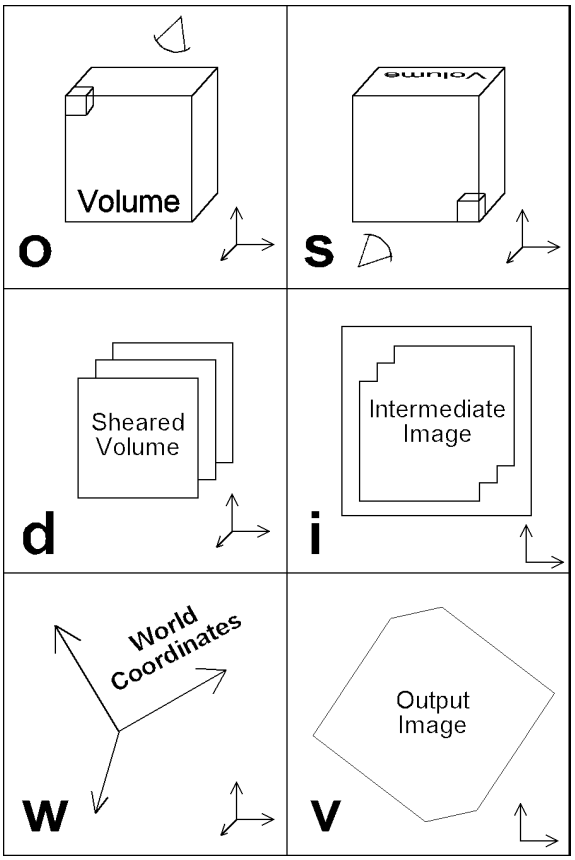


Figure 1: Coordinate systems illustrated.

$$e^o = M^{wo} * \begin{pmatrix} 0 \\ 0 \\ -1 \\ 0 \end{pmatrix} \quad (1)$$

Then the slice order and the main principal viewing axis can be determined. The main principal viewing axis determines the permutation matrix $M^{os}$, which is needed for the adaptation of the coordinate system to the three principal viewing axis aligned data sets. Slice order and permutation matrix allow the compositing step to always process the slices front-to-back with memory-aligned voxel data.

The standard object space eye position is:

$$e^s = M^{os} * e^o$$

Now we can compute the shear to deformed space:

$$M^{sd} = \begin{pmatrix} 1 & 0 & -\frac{e^s_x}{e^s_z} & 0 \\ 0 & 1 & -\frac{e^s_y}{e^s_z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{e^s_w}{e^s_z} & 1 \end{pmatrix}$$

The sheared object is scaled to the size of the intermediate image by the scaling matrix $M^{scale}$. The scaling factor depends on the object- and voxel-space volume dimensions, and on the slice order. Section 3.1 will show how to modify this matrix to control compositing speed.

The deformed and scaled object is projected to the intermediate image by:

$$M^{di} = \begin{pmatrix} 1 & 0 & 0 & \frac{width}{2} \\ 0 & 1 & 0 & \frac{height}{2} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$Width$ and $height$ are the dimensions of the intermediate image in pixels. The object is always projected to the middle of the intermediate image. The size of the image is set large enough to suit a reasonable viewing range. If this range is exceeded, the scaling matrix is adjusted so that the object fits.

The above adds up to the overall shear matrix:

$$M^{oi} = M^{di} * M^{scale} * M^{sd} * M^{os}$$

The warp matrix follows from the goal of $M^{ov} = M^{iv} * M^{oi}$, incorporating the above equations or their inverses, respectively:

$$M^{iv} = M^{wv} * M^{ow} * M^{so} * M^{ds} * (M^{scale})^{-1} * M^{id}$$

### 2.4 Permutation Of Projection And Warp

Although the permutation of the projection and the warp step is a basic premise for the perspective projection shear-warp algorithm, it has not been proved before. Our proof computes the two viewing matrices and then compares their components.

Let $P$ be the projection matrix:

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Assume $W$ is a general warp matrix:

$$W = \begin{pmatrix} w_{00} & w_{01} & w_{02} & w_{03} \\ w_{10} & w_{11} & w_{12} & w_{13} \\ w_{20} & w_{21} & w_{22} & w_{23} \\ w_{30} & w_{31} & w_{32} & w_{33} \end{pmatrix}$$

$S$ is the shear matrix:

$$S = \begin{pmatrix} 1 & 0 & e_{x,z} & 0 \\ 0 & 1 & e_{y,z} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & e_{w,z} & 1 \end{pmatrix}$$

where

$$e_{x,z} = -\frac{e^o_x}{e^o_z}; e_{y,z} = -\frac{e^o_y}{e^o_z}; e_{w,z} = -\frac{e^o_w}{e^o_z}$$

The viewing matrix is:

$$M^{ov} = P * W * S \tag{2}$$

For the shear-warp algorithm, the following matrix is used as the viewing matrix, with the projection applied before the warp:

$$V = W * P * S$$

Comparing the substantial elements of the matrices $M^{ov}$ and $V$ results in potential differences only in the first and second row of the third column:

$$M^{ov}_{02} = w_{00} * e_{x,z} + w_{01} * e_{y,z} + w_{02} + w_{03} * e_{w,z}$$

$$M^{ov}_{12} = w_{10} * e_{x,z} + w_{11} * e_{y,z} + w_{12} + w_{13} * e_{w,z}$$

and

$$V_{02} = w_{00} * e_{x,z} + w_{01} * e_{y,z} + w_{03} * e_{w,z}$$

$$V_{12} = w_{10} * e_{x,z} + w_{11} * e_{y,z} + w_{13} * e_{w,z}$$

For leading to identical results, it is sufficient that

$$w_{02} = 0 \tag{3}$$

$$w_{12} = 0 \tag{4}$$

where from (2):

$$w_{02} = M^{ov}_{00} * e_{x,z} + M^{ov}_{01} * e_{y,z} + M^{ov}_{02} + M^{ov}_{03} * e_{w,z} \tag{5}$$

$$w_{12} = M^{ov}_{10} * e_{x,z} + M^{ov}_{11} * e_{y,z} + M^{ov}_{12} + M^{ov}_{13} * e_{w,z} \tag{6}$$

Multiplying (5) and (6) by $e^o_z$ gives:

$$w_{02} * e^o_z = M^{ov}_{00} * e^o_x + M^{ov}_{01} * e^o_y + M^{ov}_{02} * e^o_z + M^{ov}_{03} * e^o_w$$

$$w_{12} * e^o_z = M^{ov}_{10} * e^o_x + M^{ov}_{11} * e^o_y + M^{ov}_{12} * e^o_z + M^{ov}_{13} * e^o_w$$

Because of (1), multiplied by $M^{ow}$ from the left side, it follows that:

$$M^{ov}_{00} * e^o_x + M^{ov}_{01} * e^o_y + M^{ov}_{02} * e^o_z + M^{ov}_{03} * e^o_w = 0$$

$$M^{ov}_{10} * e^o_x + M^{ov}_{11} * e^o_y + M^{ov}_{12} * e^o_z + M^{ov}_{13} * e^o_w = 0$$

independently of $e^o_z$, which proves (3) and (4).

Therefore, the projection and the warp matrices can be permuted.

### 2.5 Warp Complexity Comparison

Both the perspective and the parallel projection shear-warp algorithms spend most of their time in the compositing and the warp. The slightly greater number of matrix computations for the factorization in the perspective algorithm can be neglected.

In the case of parallel projection, the warp is an affine operation compared to the perspective projection warp, which is non-affine.

Let $W_{par}$ describe the general parallel projection warp matrix. Constant elements are listed as their values, $a_{rc}$ represents variable elements:

$$W_{par} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{pmatrix}$$

Multiplying $W_{par}$ by a vector $(x, y, 1)^T$ requires 4 multiplications and 4 additions, adding up to 8 floating point operations.

$W_{per}$ describes the general perspective projection warp matrix:

$$W_{per} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

In this case a multiplication with a vector $(x, y, 1)^T$ requires 6 multiplications, 6 additions, and 2 divisions, which add up to 14 floating point operations.

From this it follows that the perspective warp takes almost twice as long as the parallel warp on a system which has equal execution times for the mentioned operations.

## 3 ALGORITHMIC ISSUES

The application of the perspective shear-warp in virtual environments raises several issues which had to be solved. They are addressed in the following sub-sections.

### 3.1 Compositing

Keeping the frame rate close to constant is one of the requirements to establish and to sustain immersion in a VE. For the same reasons, it is crucial for the frame rate not to drop below a certain value, which is usually in the range of 5 to 10 frames per second, depending on the application. For the shear-warp algorithm, one way to increase rendering speed is to reduce the sampling rate.

When using texture hardware accelerated volume rendering techniques, a constant frame rate can be accomplished, e.g., by reducing the number of textures drawn [14], which leads to a reduction of the sampling rate in one dimension. Care has to be taken for the opacity correction of the remaining volume slices.

Using the shear-warp algorithm, the following approaches can be applied to increase rendering speed by a reduced sampling rate:

- Reduction of the number of slices drawn: In the compositing step, a certain number of slices are skipped, just as in the above described texture based approach. Also, the opacity values need to be corrected, which does not even slow down the rendering process, since the shear-warp algorithm already uses a look-up table for the mapping of RGBA to scalar values. The disadvantage is that the voxels of the skipped slices do not contribute to the final image. Furthermore, stepwise changes in the number of slices drawn are irritating to the user, which was shown in [13].

- Reduction of the intermediate image size: Drawing a smaller intermediate image than the optimum 1:1 pixel to voxel ratio for the first slice requires less voxels to be composited, so that rendering speed increases. The resulting image looks blurred due to the reduced intermediate image size, but due to a footprint based interpolation, no original data values are ignored.

Due to its smooth variability in the range of image pixels, we implemented the second solution for runtime frame rate adaption. Using this technique, there are no abrupt changes in image quality which could disturb the effect of immersion. Furthermore, if rendering hardware permits, rendering quality can arbitrarily be increased by enlarging the intermediate image. Figure 2 shows the effect of different intermediate image sizes on a 128x128x55 engine data set (for data source see section 6).

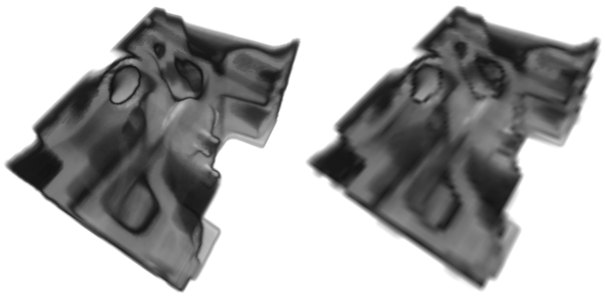

Figure 2: Intermediate image size: $2000^2$ (left) and $250^2$ (right).

Algorithmically, the adaption was implemented by modifying the parameters of matrix $M^{scale}$ (see section 2.3), thus directly affecting the size of the intermediate image. The fact that also a magnification of the intermediate image is allowed requires the compositing to provide not only footprint based resampling, but also bilinear resampling for the case that there are multiple pixels to be drawn for each voxel in a slice.

### 3.2 Warp

Section 2.5 derived the higher complexity of the perspective warp compared to the parallel warp. Since the warp matrix is not affine anymore in the case of perspective projection, the warp accounts for a more substantial part of the total rendering time, compared to the parallel warp.

Considering that the warp applies a transformation matrix to 2D data, it can be performed by 2D texturing hardware, just as the parallel projection warp is performed by Pfister's VolumePro board [10]: The OpenGL model/view matrix is set to the warp matrix, the OpenGL projection matrix is set to the identity matrix. In this case the warp matrix is not inverted, while the software warp uses its inverse to get pixel values from the intermediate image. The texturing hardware can perform the warp very fast, bilinear interpolation is added at no cost, and the final image size practically does not affect rendering speed. Furthermore, only the intermediate image has to be transferred to the graphics hardware, instead of the final image, which usually is the larger one for VE applications. On a typical SGI Onyx2 system the rendering time share of the warp is less than 2% using this method (see section 4), so that the warp time can be neglected when determining overall rendering speed.

Using texture mapping hardware for the warp does not break with our idea of a software based rendering algorithm. The advantages of the shear-warp algorithm, like easy parallelization and limitation of the volume size to fit to main memory instead of graphics hardware memory, still persist.

### 3.3 Clipping Plane

For users working with volume rendering, it is nice to have one or more arbitrarily located clipping planes to look inside of objects, if adjusting the opacity transfer function does not suffice. Texturing hardware based volume rendering makes use of the hardware accelerated clipping planes provided by OpenGL.

Shear-warp based algorithms cannot make use of the OpenGL clipping planes because they composite a 2D image with no depth information. Thus, the clipping planes have to be introduced in the compositing step. Yen et al. [16] extract thin slabs out of the volume, but the core of their approach can be applied to arbitrarily oriented clipping planes similarly: The compositing loops have to be limited to the intersections of the clipping plane. This technique can be applied similarly to both the parallel and the perspective projection algorithm. For an example see figure 3.

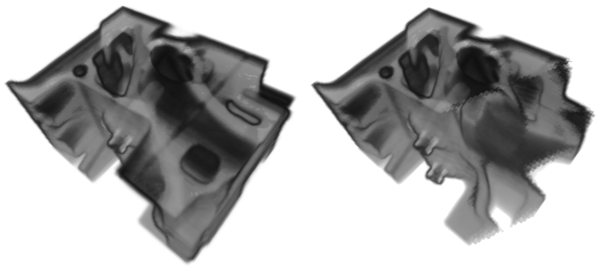

Figure 3: Engine data set: complete (left) and clipped (right).

### 3.4 Viewing Angle

Lacroute describes that if there is more than one principal viewing axis, the volume has to be subdivided into up to six separately

rendered pyramidal sub-volumes. This would impose a major performance degradation on the implementation, because several sub-volumes would have to be rendered and assembled seamlessly.

We examined this issue for the special case of a CAVE-like environment. Due to the specific geometry of the setup, all viewing rays deviate less than 90 degrees from the corresponding projection axis (see figure 4). The case of approaching 90 degrees, when image distortion would become a problem, is the case of being very close to a wall. That is not the case in typical VE situations with a tracked presenter surrounded by a non-tracked audience. Coming close to a wall is typically correlated with a viewing direction nearly perpendicular to a wall. As also discussed in the region of interest related literature (e.g., [15]), the edges of the field of view outside the region of interest can be displayed rather coarsely.

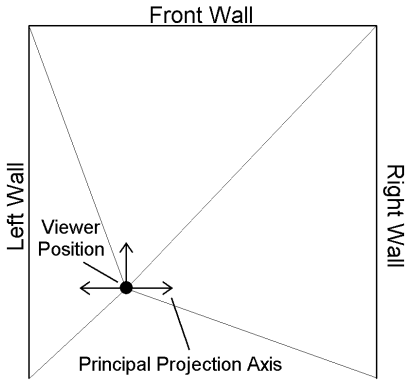


Figure 4: Multiple principal viewing axes.

### 3.5 Viewpoint

In a VE, the user needs to be free in choosing his position to look at the scene, and he can always only see things which are located within the viewing frustum. For the volume location, three cases may occur:

1. The volume is located entirely in front of the viewer.
2. The volume is located entirely behind the viewer.
3. A part of the volume is located in front of and another part is behind the viewer.

In order to find the appropriate case, a bounding box check needs to be done on the volume boundaries. In the first case, no further action is necessary, because the viewing frustum clipping is performed by the 2D warp. In the second case, the volume simply is not drawn at all. For dealing with the third case, we set a clipping plane directly in front of the user's eye point, at the location of the viewing frustum near plane, with its normal facing to the user. Thus, the user can see the data set from inside.

### 3.6 Concurrent Display Of Polygons

The current implementation of the perspective projection algorithm provides two techniques to warp the intermediate image to the screen:

- Computing the final 2D image using a pure software algorithm and pasting it to the viewport.
- Having the 2D texturing hardware perform the warp.

The first alternative does not allow for automatic object ordering; the programmer can only choose to draw the final image at a certain point of time during rendering of the scene. Since depth sorting the polygons is usually prohibitive due to their large number, the programmer's only choice is to draw the volume before or after the polygons.

The second alternative automatically provides a reasonable solution because the warp matrix transforms the intermediate image into 3D space, which corresponds roughly with the correct volume position. Thus, due to the Z buffer, the hardware draws the scene's polygons to the correct depth, relative to the volume. Only for polygons intersecting the volume, the result is not correct. Section 5 mentions an approach to overcome this drawback.

## 4 RESULTS

For the measurements of computation times, the following two systems were used:

- PC: A single processor PC with a 1.4 GHz Pentium 4, running Windows 2000, and equipped with a 3Dlabs Wildcat II 5110 graphics board.
- Onyx2: An SGI Onyx2 with 4 IR2 pipes and 14 R10000/195 MHz processors running IRIX 6.5. The tests used only one processor and were done on the monitor in mono mode.

In all tests, bilinear interpolation was used. It was rendered in RGB colors, the alpha values were defined by a linearly increasing opacity ramp.

### 4.1 Rendering Time

Figure 5 shows the rendering times (compositing and warp) for different intermediate image sizes using the adaptive perspective projection algorithm, which was described in section 3.1. This test was done on the PC with an output image size of $300^2$ pixels and a 64x64x27 voxels engine data set. Jumps occur at intermediate image edge lengths of 512 and 1024 pixels because the texturing hardware, which is used for the warp, requires image sizes which are powers of two.

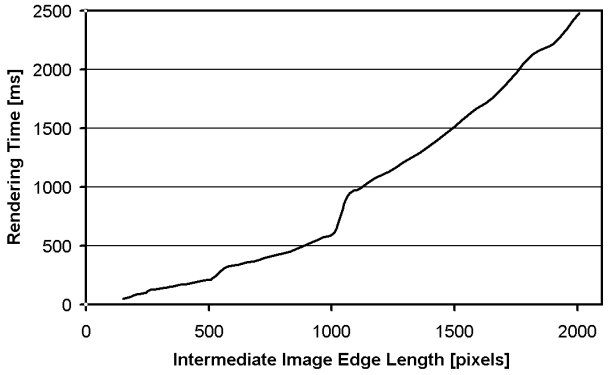


Figure 5: Rendering speed relative to intermediate image size.

### 4.2 Perspective Vs. Parallel Projection

Table 2 shows the ratio between the computation times of the perspective and the parallel projection algorithm. The output window

size was $300^2$ pixels, and the texture based warp was used. Again, the engine data set was rendered. It can be seen that, in this example, the perspective projection is about 45% slower than the parallel projection.

Table 2: Perspective vs. parallel projection.

| | PC | Onyx2 |
|---|---|---|
| Perspective : parallel projection | 1.43 | 1.46 |

### 4.3 Software Vs. Texture Based Warp

In table 3 the computation times of the two warp implementations for the perspective algorithm are given for different output image sizes and a $1024^2$ intermediate image. On both systems, the software warp is faster than the texture based warp for small images due to texturing hardware overhead.

Table 3: Software vs. texture based warp.

| Warp type | Output image size | PC [ms] | Onyx2 [ms] |
|---|---|---|---|
| Software | $256^2$ | 20 | 32 |
| Texture | $256^2$ | 30 | 68 |
| Software | $512^2$ | 60 | 108 |
| Texture | $512^2$ | 30 | 68 |

### 4.4 Compositing Vs. Warp

In table 4 the computation time ratio between compositing and warp of the perspective algorithm is shown for both the software based and the texture based warp algorithm. The Brainsmall data set was used (see section 6), the intermediate image size was $1024^2$. The table demonstrates that using texturing hardware the warp only accounts for 2 to 5% of the total rendering time, and that it is independent of the window size. In contrast, the computation time of the software based warp may get into the range of compositing time for large output image sizes.

Table 4: Compositing vs. warp.

| Warp type | Output image size | PC | Onyx2 |
|---|---|---|---|
| Software | $256^2$ | 92.1 | 9.71 |
| Texture | $256^2$ | 17.5 | 50.1 |
| Software | $512^2$ | 36.8 | 2.40 |
| Texture | $512^2$ | 17.5 | 50.1 |

### 4.5 Application To An Immersive VE

Our immersive VE, the CUBE (see figure 6) at the University of Stuttgart, is driven by the above mentioned Onyx2. Our parallelized implementation of the algorithm renders the 64x64x27 voxels engine data set at about 10 frames per second.

## 5 CONCLUSIONS AND FUTURE WORK

We have presented the application of the perspective projection shear-warp algorithm to virtual environments. We have mathematically proved an important basis for the algorithm, and we have discussed implementation issues.

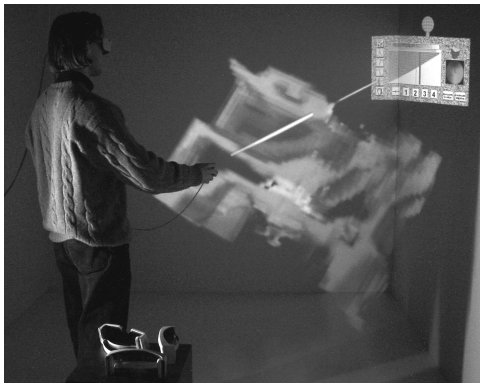

Figure 6: Shear-warp rendering in the CUBE.

A further improvement of our perspective projection algorithm would first of all mean to implement all of the features described by Lacroute: a look-up table based shading technique, shadows, and a min-max octree. In order to increase rendering speed, the compositing step needs to be parallelized for multi-processor machines.

An interesting approach to improve stereo rendering speed is given in [5], where similarities are utilized to render the stereo images. This could be adapted to the perspective projection shear-warp algorithm.

An important development would be to integrate the shear-warp algorithm with polygon-based techniques. In order to achieve a correct concurrent display for opaque polygons, the polygons would have to be rendered first. Then an inversely warped Z buffer could be used to limit the viewing rays in the shear-warp's compositing step.

## 6 ACKNOWLEDGMENTS

This work has been supported by the Deutsche Forschungsgemeinschaft within the SFB 382. The integration of the shear-warp algorithm into COVER was developed in collaboration with Uwe Wössner of the HLRS. The engine data set can be found at http://wwwvis.informatik.uni-stuttgart.de/~engel/index3.html. The Stanford Brainsmall data set is part of the VolPack volume rendering library at http://www-graphics.stanford.edu/software/volpack/.

## References

[1] K. Akeley. *RealityEngine Graphics*. ACM Computer Graphics (SIGGRAPH '93 Proceedings), pp. 109–116, 1993.

[2] B. Chen, F. Dachille, and A.E. Kaufman. *Forward Image Warping*. IEEE Visualization '99 Proceedings, IEEE Computer Society Press, pp. 89–86, 1999.

[3] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti. *Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE*. Computer Graphics(SIGGRAPH '93 Proceedings), pp. 135–142, 1993.

[4] B. Csebfalvi. *Fast Volume Rotation using Binary Shear-Warp Factorization*. Eurographics Data Visualization '99 Proceedings, pp. 145–154, 1999.

[5] T. He and A. Kaufman. *Fast Stereo Volume Rendering*. IEEE Visualization '96 Proceedings, 1996.

[6] G. Knittel and W. Strasser. *Vizard - Visualization Accelerator for Real-Time Display*. Proceedings of SIGGRAPH/Eurographics Workshop on Graphics Hardware, ACM Press, pp. 139–147, 1997.

[7] P. Lacroute. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation.* Doctoral Dissertation, Stanford University, 1995.

[8] P. Lacroute. *Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization.* IEEE Parallel Rendering Symposium '95 Proceedings, pp. 15–22, 1995.

[9] P. Lacroute and M. Levoy. *Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation*. Computer Graphics Vol. 28 (SIGGRAPH '94 Proceedings), pp. 451–457, 1994.

[10] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. *The VolumePro Real-Time Ray-Casting System.* Computer Graphics (SIGGRAPH '99 Proceedings), ACM Press, pp. 251–260, 1999.

[11] H. Pfister and A. Kaufman. *Cube-4 - A Scalable Architecture for Real-Time Volume Rendering*. ACM/IEEE Symposium on Volume Visualization '96, pp. 47–54, 1996.

[12] D. Rantzau, K. Frank, U. Lang, D. Rainer, and U. Woessner. *COVISE in the CUBE: An Environment for Analyzing Large and Complex Simulation Data*. Proc. 2nd Workshop on Immersive Projection Technology (IPTW), 1998.

[13] J. Schulze-Doebold, U. Woessner, S.P. Walz, and U. Lang. *Volume Rendering in a Virtual Environment*. Proceedings of 5th IPTW and Eurographics Virtual Environments, Springer Verlag, 2001.

[14] M. Weiler, R. Westermann, C. Hansen, K. Zimmermann, and T. Ertl. *Level-Of-Detail Volume Rendering via 3D Textures*. IEEE Volume Visualization 2000 Proceedings, 2000.

[15] R. Westermann, L. Kobbelt, and T. Ertl. *Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces*. The Visual Computer, Vol. 15, pp. 100–111, 1999.

[16] S.Y. Yen, S. Napel, and G.D. Rubin. *Fast Sliding Thin Slab Volume Visualization*. Symposium on Volume Visualization '96 Proceedings, ACM Press, pp. 79–86, 1996.