A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids

Stefan Roettger and Thomas Ertl

Visualization and Interactive Systems Group *
University of Stuttgart

Abstract

In the area of volume visualization the cell projection technique is used widely to display unstructured tetrahedral grids. In this field, the pre-integration of the ray integral has proven to be a powerful method for the display of tetrahedral grids with high quality and at interactive frame rates. Besides the actual performance of the projected tetrahedra algorithm of Shirley and Tuchman, the degree of interactivity is also limited by the update rate of the pre-integrated ray integral and the performance of the graphics card at high screen resolutions. In order to widen those two bottlenecks, we propose a hardware-accelerated pre-integration approach and a rendering method which applies standard 2D texture mapping instead of the previously used 3D texturing. As a result, the update rate of the transfer function is increased by almost a factor of hundred and the rasterization of the tetrahedra is sped up by up to a factor of three without degrading accuracy. As a side effect, pre-integrated unstructured volume rendering can be utilized on a broader range of graphics platforms, since the support of 2D texture mapping is much more common. In summary, our two-step approach greatly increases the interactivity of unstructured pre-integrated volume rendering allowing for a much wider area of application.

1 Introduction and Related Work

In order to display unstructured tetrahedral grids, the cell projection technique is used very widely in the area of volume visualization. In the literature many other methods are known for volume rendering [Drebin et al. 1988] of unstructured tetrahedral grids such as ray tracing [Kajiya 1984], scan line algorithms [Westermann and Ertl 1997; Farias et al. 2000] or hardware accelerated architectures [King et al. 2001]. In this paper, however, we will concentrate solely on enhancing the projected tetrahedra (PT) algorithm of Shirley and Tuchman [Shirley and Tuchman 1990; Wilhelms and van Gelder 1991; Williams et al. 1998], which although published more than ten years back still has great potential for improvement.

The pre-integration technique [Max et al. 1990; Roettger et al. 2000] has contributed significantly to the PT algorithm. By means of pre-integrating the three-dimensional ray integral high quality visualizations of unstructured tetrahedral grids can be achieved. Previous volume rendering approaches based on the PT algorithm were only able to support linear transfer functions [Stein et al. 1994] without introducing severe artifacts. Actual pre-integrated volume rendering approaches can cope with arbitrary transfer functions. For this purpose, the pre-integrated ray integral is stored in a 3D texture which can be mapped efficiently onto the tetrahedra using graphics hardware [Roettger et al. 2000] (see also [Engel et al. 2001] as an example of pre-integrated volume rendering on regular meshes).

However, the numerical pre-integration of the ray integral consumes a fair amount of preprocessing time. Thus, interactive updates of the transfer function are not yet possible in order to explore unstructured tetrahedral data sets comfortably. As a first step towards more interactive volume rendering of tetrahedral grids, we present a new technique which is able to recalculate the pre-integrated ray integral at interactive rates. In comparison to standard software pre-integration, a significant speed up is achieved by calculating all the colors and opacities of one 3D texture slice in parallel using the graphics hardware. Therefore, the approach is called hardware-accelerated pre-integration.

As a second step towards improved interactivity, we show how to speed up the the rasterization of the projected tetrahedra, which becomes the main limiting factor at high screen resolutions. The previously utilized 3D texture mapping approach [Roettger et al. 2000] suffers from a reduced rasterization performance, since most graphics hardware is optimized for 2D texture mapping only. In order to account for this, we transform the three-dimensional ray integral into a cylindrical coordinate system. In this system the rasterized pixels of each single tetrahedron lie on a plane going through the main cylinder axis. These planes can be easily arranged into a set of 2D textures by effectively resampling the pre-integrated ray integral in the cylinder coordinate system. Without loss of accuracy, the tetrahedra can now be rendered using the much faster 2D texture mapping. As a side effect, the method is compatible with almost every graphics accelerator available today. Thus, a much broader range of graphics platforms is supported than with 3D texture mapping.

2 Cell Projection

In this section we give a brief summary of the cell projection technique which is the basis of our new proposed methods. Basically, an unstructured tetrahedral mesh is visualized by sorting each tetrahedral cell with respect to visibility. After that, the cells are composed (blended) in a back to front fashion. Efficient visibility sorting algorithms have been studied in depth in the literature. As a starting point, we refer the reader to [Williams 1992] and [Comba et al. 1999]. For the remainder of this paper, it is supposed that the mesh is already sorted with respect to visibility. Then each cell is pro-

* http://wwwvis.informatik.uni-stuttgart.de

jected into screen space coordinates resulting in either three or four triangles depending on the orientation of the tetrahedron (see Figure 1). These projected triangles are placed around a "thick" vertex in the center. They can be rendered and composed efficiently using standard graphics hardware, which is the main advantage of the algorithm.

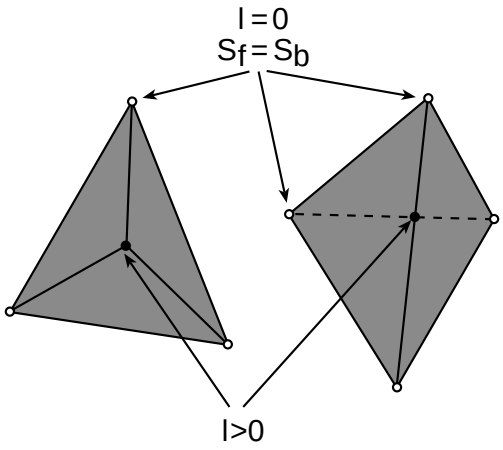

Figure 1: Decomposition of a projected tetrahedron into either three or four triangles placed around a "thick" vertex in the center (black dot). The meaning of the parameters S_f, S_b, and l is illustrated in Figure 2.

Using the volume density optical model of Williams et al. [Williams and Max 1992] (see also [Max 1995]) the color and opacity of each pixel of the projected tetrahedra is a function of the three parameters S_f, S_b, and l, which are the scalar values at the entry and the exit point of each ray segment and the length of the ray segment, respectively (see Figure 2). For a given scalar density function $f = f(x,y,z)$, the chromaticity vector $\kappa = \kappa(f(x,y,z))$ and the scalar optical density $\rho = \rho(f(x,y,z))$ are the transfer functions of this model. The three parameters S_f, S_b, and l define the domain of the three-dimensional ray integral which is given as the integrated chromaticity C_{3D} and the integrated transparency α_{3D}:

$$S_l(u) = S_f + \frac{u}{l}(S_b - S_f) \quad (1)$$

$$C_{3D}(S_f,S_b,l) = \int_0^l e^{-\int_0^v \rho(S_l(u))du} \kappa(S_l(v))dv \quad (2)$$

$$\alpha_{3D}(S_f,S_b,l) = 1 - e^{-\int_0^l \rho(S_l(u))du} \quad (3)$$

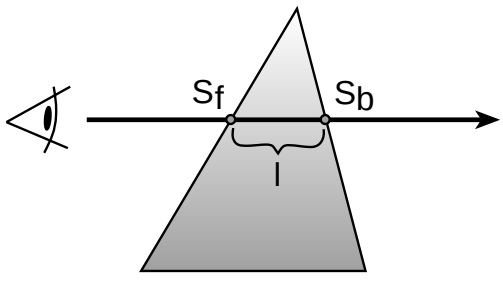

Figure 2: Intersection of the viewing ray with a tetrahedron: S_f and S_b are the scalar values of the front and back face, respectively. The length of the ray segment is denoted by l.

For a constant chromaticity vector and a linear optical density the ray integral simplifies to the solution found by Stein et al. [Stein et al. 1994]. As proposed by Roettger et al. [Roettger et al. 2000], arbitrary transfer functions can be applied by first setting up a 3D texture which contains the integrated chromaticity C_{3D} and the integrated transparency α_{3D}. Secondly, the three parameters S_f, S_b, and l serve as texture coordinates s, t, and r assigned to each vertex of the projected triangles. At the silhouette of a projected tetrahedron, for example, the length of each ray segment is zero, thus the texture coordinate r is set to zero at the silhouette vertices. Within each tetrahedron the scalar values are interpolated linearly. For obvious geometrical reasons the same is true for the length of the ray segment. Since the texture coordinates are interpolated linearly within each tetrahedron, it is straight forward to see that each pixel is rasterized with the correct colors and opacities obtained from the pre-integrated 3D texture map. For each compositing step the color I' of a pixel is calculated from the previous color I in the frame buffer using the following blend operation with back to front sorting:

$$I' := C_{3D} + (1 - \alpha_{3D}) \times I \quad (4)$$

3 First Step: Hardware-Accelerated Pre-Integration

As a first step towards more interactive pre-integrated volume rendering, we discuss how to speed up pre-integration in this section. Prior to rendering, the three-dimensional ray integral has to be computed in a preprocessing step. Analytical solutions are known for simple transfer functions such as a linear optical density. For the usual definition of the transfer function as a lookup table consisting of m entries, a time consuming numerical integration is necessary. The time spent by the numerical integration depends by an order of four on the resolution of the 3D textures as shown below. Therefore, the computation of a pre-integrated 3D texture with, for example, 256^3 voxels requires far too many operations to achieve interactive update rates with standard software integration. As a solution to this problem, we propose to utilize graphics hardware in order to speed up the pre-integration of the ray integral.

The three-dimensional ray integral is integrated numerically by iterating over the three parameters S_f, S_b and l in the range $[0,1] \times [0,1] \times [0,l_{max}]$. For each evaluation of the integral we sample the transfer function n times. The number of necessary samples should be greater or equal to the number of touched entries in the lookup table. Therefore, the total number of integration steps depends by an order of four on the resolution of the 3D texture. For each numerical integration step we use the approximation of Stein et al. [Stein et al. 1994] which assumes the chromaticity vector to be constant. Since this assumption does not hold for a linearly interpolated lookup table we oversample by a factor of o to achieve more accurate results. From empirical studies an oversampling factor of $o = 2$ is regarded to be sufficient for the described type of transfer function. Thus, the minimum number of samples per integration is $n = \lceil 2m|S_f - S_b| \rceil$. The emission and absorption of the light on its way along each ray segment is integrated numerically by repeatedly evaluating the following formula n times. The final resulting RGBA color vector is stored in a 3D texture map, which is denoted by $T_{3D}(S_f,S_b,l) = (C_{3D_{n-1}}, \alpha_{3D_{n-1}})$ with $\theta = 1 - \alpha$:

$$w = S_b + \frac{i}{n-1}(S_f - S_b), \quad i = 0,\cdots,n-1 \quad (5)$$

$$C_{3D_{i+1}} = C_{3D_i} e^{-\frac{l}{n}\rho(w)} + \frac{l}{n}\kappa(w), \quad C_{3D_0} = (0,0,0) \quad (6)$$

$$\theta_{3D_{i+1}} = \theta_{3D_i} e^{-\frac{l}{n}\rho(w)}, \quad \theta_{3D_0} = 1 \quad (7)$$

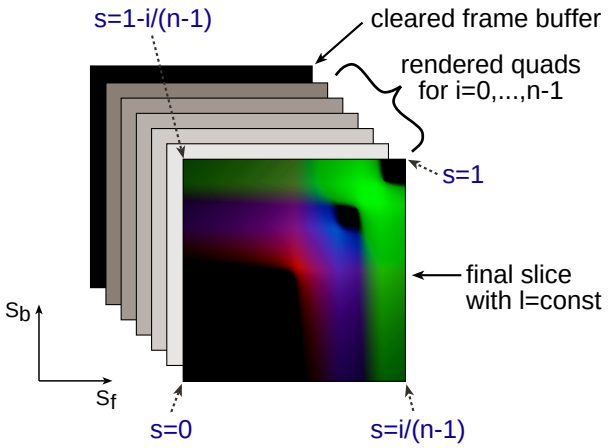

Figure 3: Hardware-accelerated pre-integration: First, the transfer function is stored in a 1D texture map. Then n rectangles are drawn and blended with the texture coordinate s assigned in the depicted fashion.

In order to speed up the numerical calculation of the ray integral, we rearrange the nesting of the four loops, such that the two innermost loops iterate over S_f and S_b. Now, texture mapping hardware can be used to substitute the two inner loops. This leads to an approach which effectively integrates the colors and opacities of one 3D texture slice in parallel. Since all voxels of a slice are integrated simultaneously we use the upper bound $n = o \cdot m$ as the number of samples. For each slice with $l = const$, we need two rendering passes to calculate both the RGB and the alpha component of the voxels. For the RGB component of the voxels, the frame buffer is initially set to $(0,0,0)$ and the transfer function is stored in a 1D texture:

$$T_{1D}(s) = (\frac{l}{n}\kappa(s), 1 - e^{-\frac{l}{n}\rho(s)}) \tag{8}$$

Then n rectangles with a size equal to the resolution of the 3D texture slices are rendered and blended into the frame buffer. The texture coordinate s is set according to Figure 3 ensuring that the RGBA color of each rendered pixel corresponds to the color at position $s = w$ of the transfer function. For the alpha component of the voxels, we proceed as described above, except that we use the primary color $(0,0,0,1)$ to modulate the 1D texture. After initializing the frame buffer to $(1,1,1)$ the transparencies θ are calculated in all three RGB channels. If the graphics hardware supports blending with a constant color then the two passes can be combined into a single one by using the constant blending color (1,1,1,0) in the source path, which results in the calculation of the transparencies in the α-channel of the frame buffer. As the final step, the contents of the frame buffer are transferred back into main memory or copied directly into a 3D texture map residing in dedicated graphics memory. In order to illustrate the progress of the hardware-accelerated pre-integration (of the RGB component), four intermediate steps from a numerical integration with a total of $n = 128$ sampling steps are shown in Figure 4. The final result can be seen on the left side of Figure 5. The applied transfer function consisted of 64 entries, which can be seen besides the images in Figure 4.

The number of integration steps and the bit depth of the frame buffer limit the accuracy of the integration. While SGI workstations support 12 bits per channel and achieve good integration quality for transfer functions of up to 256 entries (see also Table 1), current PC graphics hardware supports a maximum of 8 bits per channel. Using the pixel shaders of the NVIDIA GeForce3 or the ATI Radeon 8500 graphics accelerator several integration steps can be combined

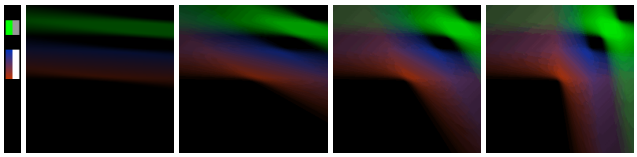

Figure 4: The RGB contents of the frame buffer after 16, 48, 80, and 112 integration steps out of a total of 128 steps. The applied transfer functions $\kappa(S)$ and $\rho(S)$ are depicted besides the images.

taking advantage of the increased internal accuracy of the graphics accelerators. In the case of the GeForce3, the three available general combiners are utilized to compute three steps of the integration with 12 bits of internal accuracy. The final combiner is configured to perform another integration step. The precision gain is therefore equivalent to reducing the number of integration steps by a factor of 4. Besides the improved accuracy, this multi-texturing approach is approximately twice as fast as single-texturing.

The RMS (root mean square) deviation of our hardware-accelerated approach from standard numerical integration is given in Table 1. Frame buffer depths of 12 bits per channel or multiple integration steps at once allow us to use transfer functions of up to 256 or 128 entries, respectively. Beyond that limit the quality of the hardware-accelerated integration is not sufficient, because quantization artifacts become visible. Transfer functions with more entries require a frame buffer depth of 16 bits per channel, which currently is supported only on SGI Octane2 and Onyx platforms. Since the quality gain achieved by oversampling is small compared to the quality loss of twice the number of sampling steps we choose $o = 1$ for hardware-accelerated pre-integration. As an example of the quantization artifacts which arise with insufficient frame buffer depths, the right image in Figure 5 shows the result of a hardware-accelerated pre-integration using 1024 integration steps and 12 bits per channel.

12 bits per chan.	RMS error	max. error
steps=32	0.1348%	13.9828%
steps=64	0.1515%	13.9661%
steps=128	0.2445%	13.8732%
steps=256	0.4565%	14.4455%
steps=512	0.9102%	19.8741%
steps=1024	1.8131%	24.0693%
8 bits per chan.	**RMS error**	**max. error**
steps=32	1.1050%	18.5028%
steps=64	2.0477%	27.3640%
steps=128	3.4929%	43.9580%
steps=256	5.0901%	70.5962%

Table 1: Integration accuracy of our hardware-accelerated pre-integration approach: The RMS error is given in comparison to standard software pre-integration. The deviations were calculated on a SGI Octane MXI with 12 bits per channel and on a PC with NVIDIA GeForce3 and 8 bits per channel for the transfer function depicted in Figure 4.

In comparison to standard numerical pre-integration our proposed hardware-accelerated pre-integration technique achieves a speedup of approximately 70-100, depending on the size of the calculated 3D texture (see Table 2). For a 3D texture with $64 \times 64 \times 32$ voxels the update rate is 5.5 Hertz on a SGI Octane MXI with 250 MHz MIPS R10K processor. Two passes for the colors and opacities are required on this platform, since the SGI Octane MXI only supports an RGB but no RGBA visual with 12 bits per channel. This limitation is not present on SGI Onyx systems.

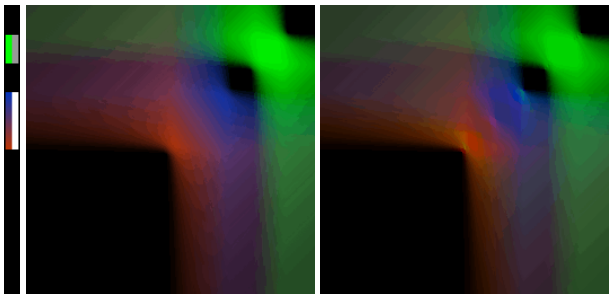

Figure 5: **Left:** The result of a pre-integration with 128 integration steps using 12 bits per channel on a SGI Octane MXI. **Right:** Quantization artifacts become visible at 1024 integration steps.

64×64×32	time	steps	speedup
software int.	13.499s	1-64	–
hardware int.	0.188s	32	71.80
128×128×64	time	steps	speed up
software int.	216.204s	1-128	–
hardware int.	2.859s	64	75.62
256×256×128	time	steps	speed up
software int.	3413.444s	1-256	–
hardware int.	35.550s	128	96.01

Table 2: Performance comparison of the standard software and the hardware pre-integration approach on a SGI Octane MXI with 250 MHz MIPS R10K processor. For comparability, the number of integration steps was chosen such that each method performs the same number of steps on the average.

4 Second Step: Cylindrical Mapping of the Ray Integral

As a second step towards more interactive pre-integrated volume rendering of unstructured grids, we propose to speed up rasterization of the tetrahedra by using a cylindrical mapping of the ray integral. For high screen resolutions the rasterization performance of the graphics card becomes a limiting factor for unstructured volume rendering. Since the approach of Roettger et al. [Roettger et al. 2000] relies on 3D textures which contain the pre-integrated ray integral, actual implementations of the algorithm are restricted to graphics hardware which supports 3D textures efficiently.

However, actual graphics hardware performs much slower when using 3D textures instead of standard 2D textures. This is due to additional texture memory accesses and a reduced texture cache coherency. In order to overcome this restriction, we seek to find a solution which utilizes 2D textures rather than the slower 3D textures. We attain this goal by first considering the following observation:

Observation: At the silhouette of each projected tetrahedron the length of the ray segments is zero and the scalar values at the front and the back face are equal (compare Figure 1).

For this reason, the rasterized pixels of the silhouette lie on the front diagonal of the pre-integrated 3D texture (see Figure 6). For the same reason, all rasterized pixels of the tetrahedron lie on a plane defined by the front diagonal and the parameters S_f, S_b, and $l > 0$ assigned to the thick vertex of the projected tetrahedron.

For convenience, we shear the texture coordinate system such that the front diagonal becomes the vertical axis of our new coordinate system defined by the base vector $(S_f - S_b, S_b, l)$ (see Figure 7). Next, the new coordinate system is transformed into cylinder coordinates with the front diagonal being the cylinder axis and $\alpha \in [0°, 180°]$ being the rotational parameter. In this coordinate system

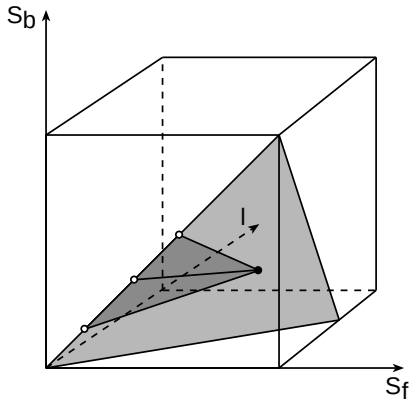

Figure 6: The rasterized pixels (dark grey) of a projected tetrahedron lie on a plane (light grey) defined by the front diagonal and the parameters S_f, S_b, and l assigned to the thick vertex (black dot).

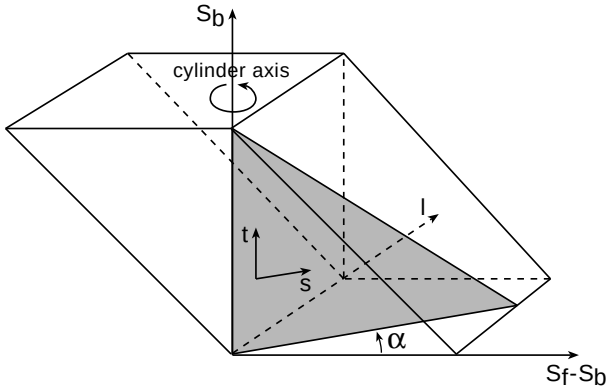

Figure 7: Sheared cylindrical representation of the ray integral: All rasterized pixels of a tetrahedron lie on a cylinder plane (grey triangle) with $\alpha = const$.

the rasterized pixels of a tetrahedron lie on a plane with $\alpha = const$. For each projected tetrahedron the three-dimensional ray integral is reduced to a two dimensional integral in sheared cylinder coordinates.

Instead of computing a two-dimensional table for each projected tetrahedron, we resample the ray integral at equidistant values of α and reconstruct the exact colors and opacities of each cylinder plane by linear interpolation of the two nearest resampled planes. Each of these resampled planes is stored as a 2D texture map in dedicated graphics memory. The linear interpolation of the resampled planes can be performed efficiently by the graphics hardware using two-pass multi-texturing.

Since the 2D textures have different size and shape for different values of α, a lot of texture memory would be wasted by scaling up to the next power of two. Also, costly trigonometric evaluations would be required for the mapping to the cylinder coordinate system. Instead, we choose a similar "cylindrical" mapping, which produces equally sized textures by orthogonally projecting the cylinder planes onto the nearest plane defined by $\alpha = 0°, 90°$, or $180°$. Assuming normalized texture coordinates $(\alpha, s, t, S_f, S_b, l \in [0,1])$, the pre-integrated ray integral is resampled into a set of square 2D textures with the same resolution as the 3D texture using the following mapping:

$$
\begin{aligned}
f : (\alpha, s, t) \mapsto & \quad (S_f, S_b, l) \\
\alpha \in [0, 0.25] : & \quad (t+s, t, 4\alpha s) \\
\alpha \in [0.25, 0.5] : & \quad (t+4(0.5-\alpha)s, t, s) \\
\alpha \in [0.5, 0.75] : & \quad (t-4(\alpha-0.5)s, t, s) \\
\alpha \in [0.75, 1] : & \quad (t-s, t, 4(1-\alpha)s)
\end{aligned} \tag{9}
$$

While rendering the tetrahedra, the parameters S_f, S_b and l have to be transformed into "cylinder" coordinates in order to choose the appropriate two textures next to α and to set the 2D texture coordinates s and t at the projected vertices. At the silhouette vertices the texture coordinates are just $(0, S_b)$. The additional overhead for the "thick" vertex is at most one division, one multiplication, and two subtractions:

$$
\begin{aligned}
f^{-1} : (S_f, S_b, l) \mapsto & \quad (\alpha, s, t) \\
S_f - S_b >= 0 \wedge l <= S_f - S_b : & \quad (\tfrac{l}{4(S_f - S_b)}, S_f - S_b, S_b) \\
S_f - S_b >= 0 \wedge l > S_f - S_b : & \quad (0.5 - \tfrac{S_f - S_b}{4l}, l, S_b) \\
S_f - S_b < 0 \wedge l >= S_b - S_f : & \quad (0.5 + \tfrac{S_b - S_f}{4l}, l, S_b) \\
S_f - S_b < 0 \wedge l < S_b - S_f : & \quad (1 - \tfrac{l}{4(S_b - S_f)}, S_b - S_f, S_b)
\end{aligned} \tag{10}
$$

In order to reconstruct the original pre-integrated 3D table exactly, the number of "cylinder" planes must be four times the resolution of the 3D texture. In most cases, however, the number of planes can be reduced by a factor of two or four without introducing visible artifacts. In the latter case the memory consumption of the 2D textures is the same as the memory consumption of the original 3D texture. By combining two of the triangular shapes into a single quadrilateral the texture memory consumption could be reduced further.

3dfx	3D texture	2D multi-texture	speedup
Voodoo3 2000	–	8.9 fps	–
NVIDIA	**3D texture**	**2D multi-texture**	**speedup**
GeForce2 MX	–	33.8 fps	–
GeForce3	47.6 fps	79.0 fps	66%
ATI	**3D texture**	**2D multi-texture**	**speed up**
Rage 128 M3	–	5.5 fps	–
Radeon 7200	13.0 fps	30.6 fps	235%
Radeon 8500	33.8 fps	38.2 fps	13%
SGI	**3D texture**	**2D texture**	**speedup**
Octane MXI	8.5 fps	12.7 fps	49%
Octane2 V8	11.4 fps	32.2 fps	282%
Onyx 3000	11.3 fps	24.1 fps	213%

Figure 8: Performance measurements: 3D texturing versus two-pass 2D multi-texturing (single 2D texture with nearest neighbour interpolation on SGI platforms).

For each rendered tetrahedron the α-parameter is usually distinct. Switching to the appropriate textures for each tetrahedron causes significant overhead for texture state management. A solution to minimize this overhead is to arrange the resampled textures into one large tiled texture. In Figure 9 eight slices of a pre-integrated 3D texture are displayed for an example transfer function which consisted of four interleaved orange and blue regions with constant opacity. The corresponding tiled 2D texture is given in Figure 10.

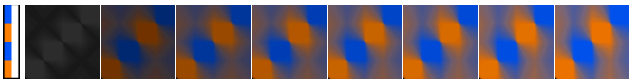

Figure 9: Eight slices with increasing ray segment length l from a pre-integrated 3D texture with 64^3 voxels. The transfer function is shown besides the images.

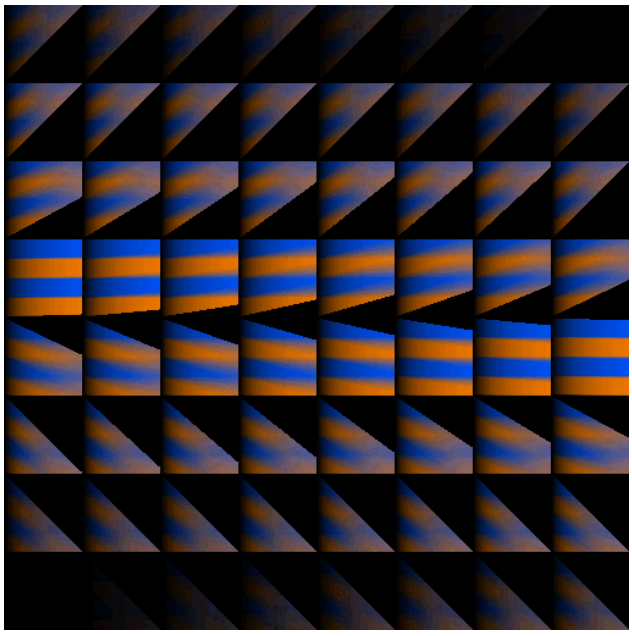

Figure 10: Tiled "cylindrical" representation of the pre-integrated 3D texture shown in Figure 9 using 64 "cylinder" planes with a tile size of 64^2 pixels. The tile in the left bottom corner corresponds to $\alpha = 0^\circ$ and the tile in the right top corner to $\alpha = 180^\circ$.

By means of the described two-pass multi-texturing approach pre-integrated unstructured volume rendering can be utilized on a much broader range of graphics platforms. For example, it can be utilized on an Apple iBook with Rage 128 Mobility M3 graphics chip which has no support for 3D texturing (see Figure 12 which shows a bonsai [Roettger and Thomandl 2000] rendered on an Apple iBook).

On platforms that support 3D texturing the rasterization performance is improved by up to an factor of three. The experimental measurements in Table 8 have been conducted by rendering a spherical distance volume (depicted in Figure 11) with both the 3D texture from Figure 9 and the two-dimensional analogue from Figure 10. The distance volume consisted of only 2560 tetrahedra so that the rasterization of the tetrahedra was the main limiting factor. This was true even for the comparably small window size of 512×384 pixels.

5 Conclusion

The described enhancements widen the two main bottlenecks of pre-integrated volume rendering of unstructured grids, namely the slow update rate of the pre-integrated ray integral and the poor rendering performance at high screen resolutions. As a consequence of the proposed hardware-accelerated numerical pre-integration, the transfer function can be manipulated interactively to explore unstructured data sets comfortably. In comparison to previous approaches the update rate was sped up by a factor of almost 100.

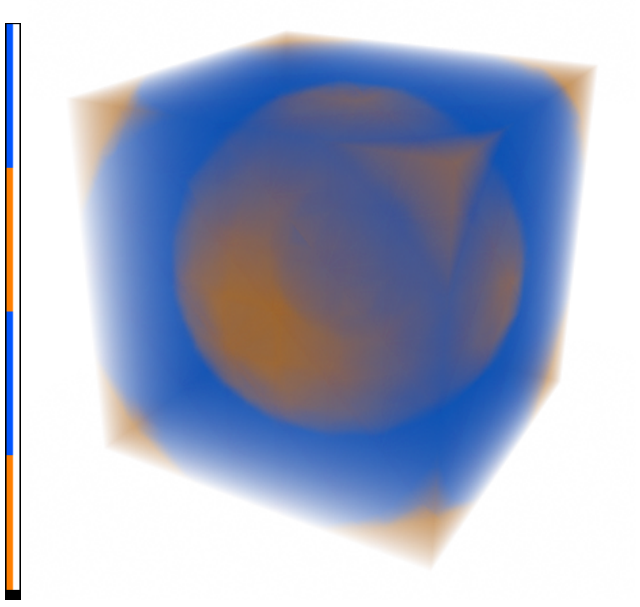

Figure 11: Spherical distance volume rendered with an interleaved orange and blue transfer function which is shown besides.

The rasterization performance, which is the main bottleneck at high screen resolutions, was increased significantly by utilizing 2D multi-texturing instead of the previously applied 3D texturing. As a welcome side effect, a much broader range of graphics platforms is supported by our 2D multi-texturing approach. This is an important improvement for the practical application and the interactive usability of pre-integrated unstructured volume rendering.

References

COMBA, J., KLOSOWSKI, J. T., MAX, N. L., MITCHELL, J. S. B., SILVA, C. T., AND WILLIAMS, P. L. 1999. Fast Polyhedral Cell Sorting for Interactive Rendering of Unstructured Grids. *Computer Graphics Forum (Proc. Eurographics '99) 18*, 3, 369–376.

DREBIN, R. A., CARPENTER, L., AND HANRAHAN, P. 1988. Volume Rendering. *Computer Graphics 22*, 4, 65–74.

ENGEL, K., KRAUS, M., AND ERTL, T. 2001. High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Eurographics Workshop on Graphics Hardware '01*, ACM SIGGRAPH, 9–16.

FARIAS, R., MITCHELL, J., AND SILVA, C. 2000. An Efficient and Exact Projection Algorithm for Unstructured Volume Rendering. In *Proc. Volume Visualization Symposium '00*, ACM Press, 91–99.

KAJIYA, J. T. 1984. Ray Tracing Volume Densities. In *Proc. SIGGRAPH '84*, ACM, 165–174.

KING, D., WITTENBRINK, C., AND WOLTERS, H. 2001. An Architecture For Interactive Tetrahedral Volume Rendering. *Proc. International Workshop on Volume Graphics '01*, 101–112.

MAX, N. L., HANRAHAN, P., AND CRAWFIS, R. 1990. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *Computer Graphics (San Diego Workshop on Volume Visualization) 24*, 5, 27–33.

Figure 12: Hierarchical unstructured bonsai data set with approximately 1,200,000 tetrahedra rendered on an Apple iBook with our proposed multi-texturing technique. The transfer functions are depicted besides.

MAX, N. L. 1995. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2, 99–108.

ROETTGER, S., AND THOMANDL, B. 2000. Three Little Bonsais. *http://wwwvis.informatik.uni-stuttgart.de/~roettger/data/Bonsai*.

ROETTGER, S., KRAUS, M., AND ERTL, T. 2000. Hardware-Accelerated Volume and Isosurface Rendering Based On Cell-Projection. In *Proc. Visualization '00*, IEEE, 109–116.

SHIRLEY, P., AND TUCHMAN, A. 1990. A Polygonal Approximation to Direct Scalar Volume Rendering. *ACM Computer Graphics (San Diego Workshop on Volume Visualization) 24*, 5, 63–70.

STEIN, C. M., BECKER, B. G., AND MAX, N. L. 1994. Sorting and Hardware Assisted Rendering for Volume Visualization. In *Symposium on Volume Visualization '94*, IEEE, 83–89.

WESTERMANN, R., AND ERTL, T. 1997. The VSBUFFER: Visibility Ordering of Unstructured Volume Primitives by Polygon Drawing. In *Proc. Visualization '97*, IEEE, 35–42.

WILHELMS, J., AND VAN GELDER, A. 1991. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics 25*, 4, 275–284.

WILLIAMS, P. L., AND MAX, N. L. 1992. A Volume Density Optical Model. In *Computer Graphics (Workshop on Volume Visualization '92)*, ACM, 61–68.

WILLIAMS, P. L., MAX, N. L., AND STEIN, C. M. 1998. A High Accuracy Volume Renderer for Unstructured Data. *Transactions on Visualization and Computer Graphics 4*, 1, 37–54.

WILLIAMS, P. L. 1992. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics 11*, 2, 103–126.