

High-Quality Unstructured Volume Rendering on the PC Platform

Stefan Guthe* Stefan Roettger† Andreas Schieber† Wolfgang Strasser* Thomas Ertl†

*WSI/GRIS University of Tuebingen

†IfI/VIS University of Stuttgart

Abstract

For the visualization of volume data the application of transfer functions is used widely. In this area the pre-integration technique allows high quality visualizations and the application of arbitrary transfer functions. For regular grids, this approach leads to a two-dimensional pre-integration table which easily fits into texture memory. In contrast to this, unstructured meshes require a three-dimensional pre-integration table. As a consequence, the available texture memory limits the resolution of the pre-integration table and the maximum local derivative of the transfer function. Discontinuity artifacts arise if the resolution of the pre-integration table is too low.

This paper presents a novel approach for accurate rendering of unstructured grids using the multi-texturing capabilities of commodity PC graphics hardware. Our approach achieves high quality by reconstructing the colors and opacities of the pre-integration table using the high internal precision of the pixel shader. Since we are using standard 2D multi-texturing we are not limited in the size of the pre-integration table. By combining this approach with a hardware-accelerated calculation of the pre-integration table, we achieve both high quality visualizations and interactive classification updates.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Picture and Image GenerationGraphics processors; I.3.3 [Computer Graphics]: Picture and Image GenerationViewing algorithms.

Keywords: *Volume Rendering, Unstructured Meshes, Cell Projection, Graphics Hardware, Texture Mapping.*

1. Introduction

Due to the increasing flexibility of commodity graphics hardware the pre-integration technique has become widely available for the visualization of volume data on regular grids. Previous approaches for unstructured meshes, that is irregular tetrahedral grids, employed a 3D texture to effectively apply pre-integration. Although the resulting images are of high quality, there are several restrictions due to the limited amount of available texture memory. Transfer functions with high gradients require a high resolution pre-integration table, which does not fit easily into the dedicated texture memory. To circumvent this restriction we propose to implicitly store the 3D texture by means of multiple 2D textures. Then the colors and opacities of the three-dimensional pre-integration table can be reconstructed accurately with the high internal precision of the pixel shader.

1.1. Related Work

Direct volume rendering of unstructured meshes was dramatically accelerated by the Projected Tetrahedra (PT) algorithm of Shirley and Tuchman ^{13, 18}. In order to compensate for the limitations of the PT algorithm there exist numerous competing approaches, such as ray tracing ³, ray casting ¹⁴, slicing ²³, and sweep-plane algorithms ¹⁷.

The original PT algorithm interpolates the opacities and colors linearly between the vertices, resulting in Mach bands as reported by Max et al. ⁶. Extending the original algorithm, Stein et al. ¹⁶ presented a solution for the correct interpolation of opacities by utilizing 2D texture mapping. However, this method is restricted to linear transfer functions for the opacity and still interpolates colors linearly ignoring the transfer function inside the tetrahedra. Roettger et al. ¹¹ presented an improvement to this algorithm using pre-integrated 2D textures and a linear interpolation of the opacity. They also introduced a new tech-

nique which is based on 3D texture mapping. This technique allows the correct interpolation of both the colors and the opacities employing the pre-integration method first introduced by Max et al. ⁷. Although 3D texture mapping is available on recent PC graphics hardware, the limited amount of texture memory restricts the accuracy of the classification. Therefore classifications containing high gradients do not render acceptable results. Modern PC graphics hardware, for instance the ATI Radeon 8500 and the NVIDIA GeForce4, allow more sophisticated approaches using dependent textures, multi-texturing, per-pixel shading, and hardware accelerated pre-integration. This enables us to overcome the limited size of the three-dimensional pre-integration table.

1.2. Paper Overview

The remainder of this paper is organized as follows: In Section 2 the basic cell projection algorithm and the optical model used throughout the paper are described. A new approach for approximating the three-dimensional ray integral by means of multiple 2D textures is discussed in Section 3. An efficient hardware accelerated calculation of the ray integral is given in Section 4. Employing the new approximations, the experimental results obtained on different graphics hardware are presented in Section 5. Finally, we conclude our paper and outline future work.

2. Basic Algorithm

The Projected Tetrahedra algorithm can be outlined as follows: All tetrahedra are sorted according to their visibility (see also ^{19, 14, 15, 1}), classified to their projective profile and split into triangles as seen in Figure 1. In a more general approach each cell is split into several tetrahedra during a precomputational step. However, it is also possible to render hexahedral cells directly using the extension of Schussman and Max ¹². Although our approach is compatible with this extension, we restrict ourselves to tetrahedra throughout this paper.

While the original PT algorithm uses a linear interpolation of colors and opacities between the triangle vertices, Stein et al. suggest to use a 2D texture map for the exponential interpolation of the opacities. The texture coordinates assigned to each vertex correspond to the average extinction coefficient τ and the thickness l of the projected cell, whereas the texture map contains the α -component set to $\alpha = 1 - e^{-\tau l}$. Since the texture coordinates are interpolated linearly, this approach is restricted to linear transfer functions $\tau = \tau(f(x, y, z))$ with f being the scalar function. In their high accuracy volume renderer (HIAC) Williams et al. ²¹ extended this approach to piecewise linear transfer functions.

Assuming a linear interpolation inside the tetrahedra, it can easily be seen that f varies linearly along each viewing ray. Therefore the integrated chromaticity C and opacity α

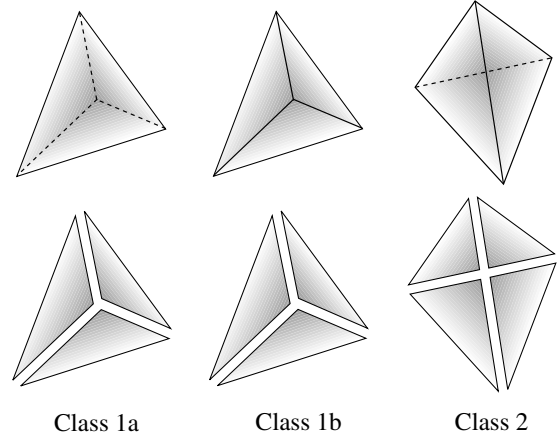


Figure 1: Classification of non-degenerated projected tetrahedra (top row) and the corresponding decomposition (bottom row) according to ¹³.

of the ray segment can be defined by a three-dimensional function depending on S_f , S_b and l (see Figure 2). Using the optical model of Williams and Max ^{20, 8, 21} with the chromaticity vector $\kappa = \kappa(f(x, y, z))$ and the scalar optical density $\rho = \rho(f(x, y, z))$, the ray integral is given as follows:

$$S_l(x) = S_f + \frac{x}{l}(S_b - S_f)$$

$$C(S_f, S_b, l) = \int_0^l e^{-\int_0^u \rho(S_l(t)) dt} \kappa(S_l(t)) \rho(S_l(t)) dt$$

$$\alpha(S_f, S_b, l) = 1 - e^{-\int_0^l \rho(S_l(t)) dt}$$

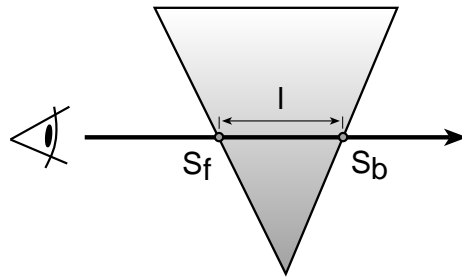


Figure 2: Intersection of a tetrahedral cell with a viewing ray. S_f and S_b are the scalar values of the front and back face, respectively; l denotes the length of the ray segment.

These integrals are computed using the numerical integration described by Engel et al. ². For each compositing step we calculate the color I' of a pixel from the previous color I by using the following blend operation:

$$I' = C + (1 - \alpha) \times I.$$

Employing the 3D texturing approach as proposed by Roettger et al. ¹¹ both κ and ρ need not be linear, thus arbitrary transfer functions can be applied. However, 3D textures require a huge amount of texture memory to achieve high quality images for arbitrary transfer functions. Beside the memory limitations of consumer graphics hardware 3D texture mapping also results in a decreased rendering performance.

3. High Resolution Ray Integral

Since high resolution 3D textures require huge amounts of texture memory, we separate the three-dimensional function of the volume density optical model. Unfortunately, only the opacity can be separated easily. The chromaticity needs to be approximated by means of a linear combination of two-dimensional functions.

3.1. Opacity Reconstruction

Since the opacity depends on the average density along the viewing ray and the length l of the ray segment, it can be separated as follows:

$$\hat{\rho}(S_f, S_b) = \int_0^1 \rho(S_f + t(S_b - S_f)) dt \quad (1)$$

$$\alpha_{1D}(x) = 1 - e^{-x} \quad (2)$$

For each rendered pixel we derive the average density $\hat{\rho}$ from a 2D texture map (Equation 1) and compute the final opacity α_{1D} by means of a 1D dependent texture lookup (Equation 2).

In order to further increase the accuracy of the reconstructed α values, the dependent texture is extended to hold the higher 8 bits of a 16 bit α value in the alpha channel A and the lower 8 bits in the additional luminance channel L . In order to map the maximum 16 bit α value to 1, it is scaled by the factor $\frac{256}{257}$. Since the resulting equation $\alpha_{1D} = \frac{256A}{257} + \frac{L}{257}$ is linear, the texture interpolation delivers a true 16 bit α lookup.

Compared to the linear approximation of the opacity using the 2D texturing approach of Roettger et al. ¹¹ the resulting images are significantly improved, as illustrated in Figure 3.

3.2. Chromaticity Reconstruction

In order to achieve a high-quality approximation of the chromaticity, we pre-integrate the normalized chromaticities $\hat{C}_l = \frac{C_l}{\alpha_l}$ for $l \rightarrow 0$ and $l = l_{max}$ with l_{max} being the maximum length of the ray segments. The normalized emission \hat{C}_0 and the difference ΔC_1 of the normalized emissions \hat{C}_0 and $\hat{C}_{l_{max}}$ are stored in two high resolution 2D textures. The latter emissions are defined as follows:

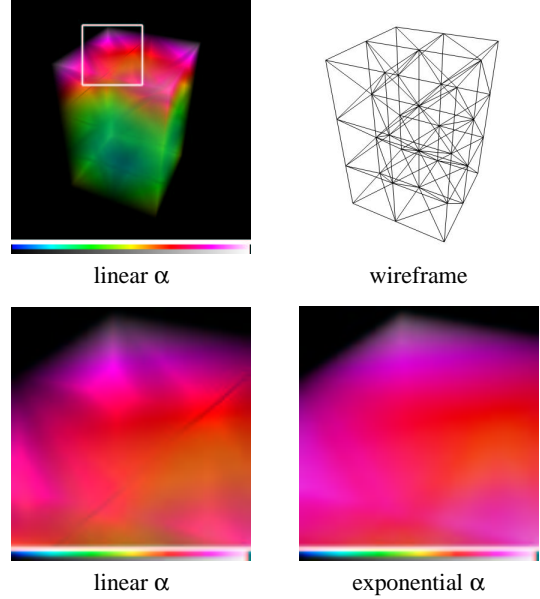


Figure 3: Comparison between linear approximation and correct exponential α . The corresponding transfer function which is split into normalized emission and opacity can be seen below each image.

$$\hat{C}_0(S_f, S_b) = \frac{\int_0^1 \kappa(S_1(t)) \rho(S_1(t)) dt}{\int_0^1 \rho(S_1(t)) dt}$$

$$\hat{C}_{l_{max}}(S_f, S_b) = \frac{C(S_f, S_b, l_{max})}{\alpha(S_f, S_b, l_{max})}$$

$$\Delta C_1(S_f, S_b) = \hat{C}_{l_{max}}(S_f, S_b) - \hat{C}_0(S_f, S_b)$$

Using the texture setup of Table 1, we implement the following approximation of the volume optical density model by utilizing dependent textures and the pixel shader on the NVIDIA GeForce4 ²⁴ and the ATI Radeon 8500 ⁹ graphics adapter:

$$C(S_f, S_b, l) \approx \hat{C}_{lin}(S_f, S_b, l) \alpha(S_f, S_b, l)$$

$$\hat{C}_{lin}(S_f, S_b, l) = \hat{C}_0(S_f, S_b) + \frac{l}{l_{max}} \Delta C_1(S_f, S_b)$$

$$\alpha(S_f, S_b, l) = \alpha_{1D}(l \rho(S_f, S_b))$$

unit	coordinates	RGB	A
0	S_f, S_b	$\hat{C}_0(S_f, S_b)$	$\rho(S_f, S_b)$
1	S_f, S_b	$\Delta C_1(S_f, S_b)$	-
2	$l \rho(S_f, S_b)$	-	$\alpha_{1D}(l \rho(S_f, S_b))$

Table 1: Texture setup for dependent texture mapping.

This is a linear approximation in l for every pair of S_f and S_b . As seen in Figure 4, the linear approximation is not accurate for transfer functions that contain high gradients. For an improved reconstruction we approximate the chromaticity by a polynomial of degree $n > 1$ in l with the coefficients $\tilde{C}_i, i = 0 \dots n$. This is similar to the polynomial texture mapping approach of Malzbender et al. ⁵, which reconstructs the colors of a surface by a biquadratic polynomial. In our case the approximated chromaticity is given by the polynomial

$$C(S_f, S_b, l) \approx \alpha(S_f, S_b, l) \sum_{i=0}^n \frac{l^i}{l_{max}^i} \tilde{C}_i(S_f, S_b).$$

To compute the polynomial coefficients \tilde{C}_i we pre-integrate the chromaticity at $l = \frac{i}{l_{max}}$ for $i = 0 \dots n$ and construct a polynomial through each of these points for every pair of S_f and S_b . This corresponds to the computation of $n + 1$ slices with $l = const$ of the pre-integration table.

Since the number of texture units is limited, we can only use a polynomial approximation with a degree of up to 2 on the GeForce4 and of up to 4 on the Radeon 8500. In the latter case the rasterization performance drops by almost 50%, but the quality of the approximation is only improved slightly. Therefore a polynomial degree of 2 should be preferred (see Figure 4). The corresponding texture setups are depicted in Table 2. The polynomial coefficients are scaled to the maximum possible texel range $[-1 \dots 1]$ to improve the precision of the approximation. Additionally, the α values are reconstructed with 16 bits of accuracy.

unit	coordinates	RGB	A
0	S_f, S_b	$\tilde{C}_0(S_f, S_b)$	$\rho(S_f, S_b)$
1	S_f, S_b	$\tilde{C}_1(S_f, S_b)$	-
...
n	S_f, S_b	$\tilde{C}_n(S_f, S_b)$	-
n+1	$l\rho(S_f, S_b)$	-	$\alpha_{1D}(l\rho(S_f, S_b))$

Table 2: Texture setup for polynomial color approximation of the three-dimensional ray integral (with a maximum polynomial degree of $n = 2$ on the GeForce4 and of $n = 4$ on the Radeon 8500).

4. Hardware Accelerated Pre-Integration

In order to visualize volume data comfortably one needs to change the transfer function interactively. Whenever the transfer function is modified the pre-integration table has to be recomputed. For a resolution of 512^2 and a polynomial degree of 4, for instance, this requires approximately 11 seconds on a Pentium 4 running at 2 GHz which is far too slow for interactive updates of the transfer function. In order to

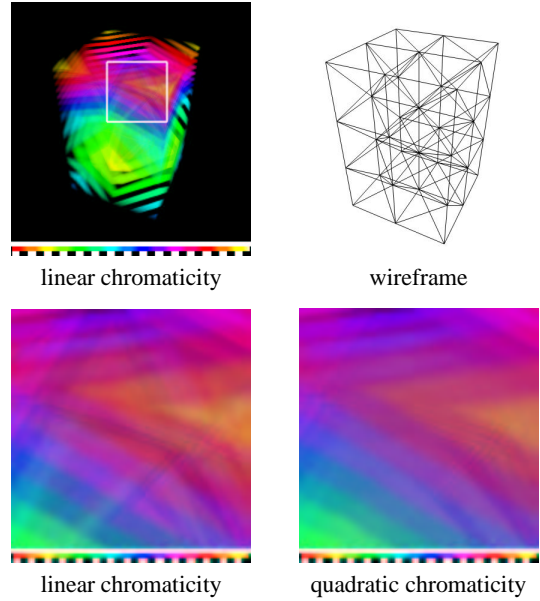


Figure 4: Comparison between linear and quadratic color approximation combined with 16 bit α , for the transfer function seen below each image.

speed up the calculation of the pre-integration table we utilize graphics hardware for the purpose of numerical integration. We maintain a high level of accuracy by using the high internal precision of the pixel shader.

The numerical integration of the ray segments is performed by sampling the integral m times. At each sampling step, the integrated chromaticity κ and the integrated opacity α are blended with the corresponding entries of the transfer function.

As described by Engel et al. ² the integrated opacity can be calculated quickly by the difference of two definite integrals. If self-attenuation is assumed to be negligible the same approach can be used to efficiently calculate the integrated chromaticities. This assumption is valid for volume slicing, since the ray segment lengths l are usually small. In the case of unstructured volume rendering, however, this assumption does not hold, thus self-attenuation cannot be neglected. As a consequence, the numerical integration of the chromaticities is not fast enough to achieve interactive updates of the transfer function. But according to Roettger et al ¹⁰, the chromaticities of one slice of the pre-integration table can be integrated in parallel by using a hardware-accelerated approach. For each slice with a constant ray segment length l this is accomplished by blending m quadrilaterals containing the sampled transfer function for every pair of S_f and S_b into the frame buffer. The sampled transfer function is reconstructed from a 1D texture (see Table 3). For this purpose, the texture coordinate s of each vertex of the quadrilaterals is assigned as shown in Figure 5.

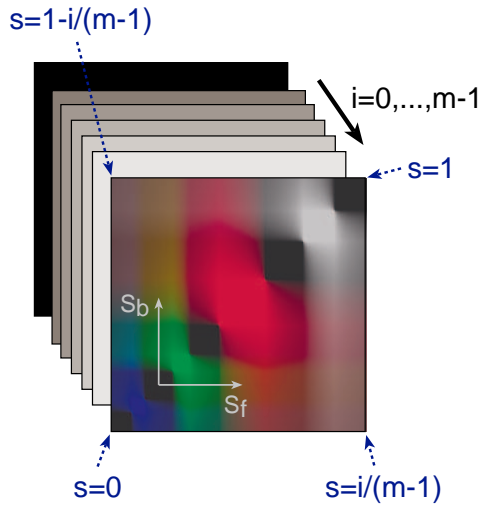


Figure 5: Texture coordinate setup for the hardware-accelerated pre-integration of one slice of the three-dimensional pre-integration table.

As the 8 bit frame buffer depth of current PC graphics hardware limits the accuracy of the numerical integration, we integrate the chromaticity with the higher internal accuracy of the pixel shader. Combining two channels of the frame buffer for each integrated color component of the chromaticity, a total accuracy of 16 bit can be achieved. In practice however, a bit depth of 12 has turned out to be sufficient.

We store the chromaticity and opacity of the transfer function for a given length l and the number of integration steps m in a 1D texture as defined in Table 3. To effectively represent high gradients in the transfer function, we construct the 1D texture with the highest possible resolution instead of using a linear interpolation of the 1D texture.

channel	meaning	value
red	high 8 bit (chromaticity)	$\kappa(s)$
green	low 4 bit (chromaticity)	$\kappa(s)$
blue	high 8 bit (opacity)	$1 - e^{-\frac{l}{m}\rho(s)}$
alpha	low 4 bit (opacity)	$1 - e^{-\frac{l}{m}\rho(s)}$

Table 3: 1D texture used for hardware-accelerated pre-integration.

On the Radeon 8500 the numerical integration is implemented using a method called ping pong filtering⁹. For each blending step an RGBA texture contains the previously integrated chromaticity in the red (high 8 bits) and alpha channel

(low 4 bits). First, the original 12 bit chromaticity is reconstructed in the pixel shader by multiplying the low bits with $\frac{1}{256}$ and adding the result to the high bits. Note that a texture entry of 255 in the high bits already represents a value of 1.0. Next, the chromaticity and opacity of the transfer function are reconstructed from the 1D texture in the same fashion. Then the chromaticity is multiplied by the opacity, the result of the previous iteration is multiplied by one minus the opacity, and the sum of both yields the new integrated chromaticity. Finally, the integrated chromaticity is split into 8 high and 4 low bits and is written back into the corresponding ping pong texture.

The Radeon 8500 masks out all bits representing values higher than 1.0 or lower than $\frac{1}{256}$. Therefore the high 8 bits are extracted automatically, whereas the low 4 bits are extracted by simply multiplying the 12 bit chromaticity with 256. In contrast to this, the GeForce4 always uses saturation logic instead of bit masking. Therefore the low 4 bits can only be extracted on the Radeon 8500.

A speedup of nearly two is achieved by performing four subsequent integration steps at once in the pixel shader. Since each RGB color component has to be computed separately, the hardware-accelerated pre-integration needs to be performed three times for every required slice of the pre-integration table. Each component of a pre-integrated slice is transferred back into main memory and recombined with the other color channels. This results in 9 pre-integration cycles for a polynomial approximation of a degree of 2, for example.

In contrast to software numerical integration, this hardware-accelerated approach allows to update the pre-integration table interactively. With respect to integration accuracy the hardware-accelerated method exhibits a higher integration error which is due to the 12 bit quantization. An example of these quantization artifacts is given in Figure 6.

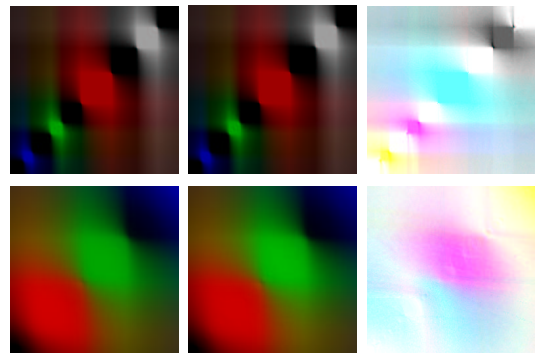


Figure 6: Comparison between hardware (left) and software (middle) pre-integration, including the error (right, scaled by a factor of 8 and inverted) for $m = 128$ sampling steps.

5. Results

In the previous chapters we have demonstrated that the multi-texturing capabilities of modern PC graphics accelerators can be utilized to bring high-quality pre-integrated volume rendering of unstructured grids to the PC platform. A comparison of the visual quality of the proposed methods is given in Figure 7. The best approximation of the pre-integration table is achieved by using 16 bits for the representation of the opacities and a polynomial of degree 4 for the reconstruction of the chromaticities. A polynomial of degree 2 is only slightly less accurate, but performs significantly faster due to reduced rasterization requirements. Because of the high internal precision of the pixel shader and the representation of the opacities with 16 bits the results are even better than those obtained with a 3D texturing setup. Using our hardware-accelerated pre-integration approach we are able to maintain high update rates of the pre-integration table. In comparison to software integration the achieved speedup is about 700% on a PC equipped with a Pentium 4 running at 2 GHz and an ATI Radeon 8500 (compare Table 4).

software	setup of textures
linear color ($n = 1$)	4.4s
polynomial $n = 2$	6.6s
polynomial $n = 4$	11.0s
Radeon 8500	setup of textures
linear color ($n = 1$)	0.6s
polynomial $n = 2$	1.0s
polynomial $n = 4$	1.7s

Table 4: Preprocessing times for 2D multi-texturing with a texture resolution of 512^2 .

The total rendering time is almost independent of the chosen reconstruction method (except for $n = 4$). It depends mainly on the sorting algorithm [19, 15, 22] and the transfer speed between the CPU and the graphics adapter (see Table 5). For comparison purposes the experimental results are given for a polynomial degree of 2. We achieve up to 600,000 tetrahedra per second depending on the sorting algorithm. Approximately half of the time is spent by sorting, while the other half is spent by rendering. The lower performance for rendering the Bucky Ball data set is due to a larger variation of the scalar values which lead to a reduced texture cache coherence.

6. Conclusion and Future Work

We presented a new approach for pre-integrated rendering of projected tetrahedra on commodity PC graphics hardware. We employed 2D multi-texturing and pixel shading to reconstruct the three-dimensional pre-integration table. Because

GeForce4	#tetra	numeric	MPVO	XMPVO
Blunt Fin	187k	3.18 fps	2.64 fps	2.35 fps
Bucky Ball	177k	2.46 fps	2.19 fps	2.05 fps
Radeon 8500	#tetra	numeric	MPVO	XMPVO
Blunt Fin	187k	2.51 fps	2.20 fps	1.99 fps
Bucky Ball	177k	2.09 fps	1.98 fps	1.87 fps

Table 5: Display times including visibility sorting on a Pentium 4 running at 2 GHz using a polynomial approximation of degree 2 and a 1280×960 view port. The applied sorting algorithms are numerical sorting [22], MPVO [19], and XMPVO [15].

of the reduced memory requirements of the employed 2D textures, our method is capable of applying high resolution transfer functions. We further presented a high quality numerical pre-integration method which utilizes the graphics hardware to decrease the classification update time. Since our approach uses the high internal precision of the pixel shader, the resulting images are of a much higher quality in comparison to the previously applied 3D texture mapping approach.

7. Acknowledgements

The authors would like to thank Martin Kraus for his ideas and discussions and Michael Dogget from ATI for providing a Radeon 8500. Part of this work has been funded by the SFB grant 382 of the German Research Council (DFG).

References

1. João Comba, James T. Klosowski, Nelson Max, Joseph S. B. Mitchell, Claudio T. Silva, and Peter L. Williams. Fast Polyhedral Cell Sorting for Interactive Rendering of Unstructured Grids. In *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 369–376, 1999. 2
2. K. Engel, M. Kraus, and Th. Ertl. High-Quality Pre-Integrated Volume Rendering using Hardware-Accelerated Pixel Shading. In *Proc. Eurographics / SIGGRAPH Workshop on Graphics Hardware '01*, Annual Conference Series, pages 9–16, 2001. 2, 4
3. James T. Kajiya. Ray Tracing Volume Densities. In *Proc. SIGGRAPH '84*, pages 165–174. ACM, 1984. 1
4. A. Kanitsar. Christmas Tree Data Set. <http://ringlotte.cg.tuwien.ac.at/datasets/XMasTree/XMasTree.html>, 2002. 8
5. Tom Malzbender, Dan Gelb, and Hans Wolters. Polynomial Texture Maps. In *SIGGRAPH 2001, Com-*

- puter Graphics Proceedings, Annual Conference Series, pages 519–528, 2001. 4
6. N. Max, B. Becker, and R. Crawfis. Flow Volumes for Interactive Vector Field Visualization. In *Proc. Visualization '93*, pages 19–24. IEEE Computer Society Press, 1993. 1
 7. N. L. Max, P. Hanrahan, and R. Crawfis. Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions. *Computer Graphics (San Diego Workshop on Volume Visualization)*, 24(5):27–33, 1990. 2
 8. Nelson Max. Optical Models for Direct Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995. 2
 9. Jason L. Mitchell. 1.4 Pixel Shaders. *Meltdown*, 2001. 3, 5
 10. Stefan Roettger and Thomas Ertl. A Two-Step Approach for Interactive Pre-Integrated Volume Rendering of Unstructured Grids. In *Proceedings of the 2002 Symposium on Volume Visualization (VOLVIS-02)*. ACM Press, 2002 (to appear). 4
 11. Stefan Roettger, Martin Kraus, and Thomas Ertl. Hardware-Accelerated Volume and Isosurface Rendering Based on Cell-Projection. In *Proc. Visualization 2000*, pages 109–116. IEEE Computer Society Technical Committee on Computer Graphics, 2000. 1, 3
 12. Greg Schussman and Nelson Max. Hierarchical Perspective Volume Rendering using Triangle Fans. In *Volume Graphics*, Proceedings of the International Workshop on Volume Graphics, pages 309–320, 2001. 2
 13. Peter Shirley and Allan Tuchman. A Polygonal Approximation for Direct Scalar Volume Rendering. In *Proc. San Diego Workshop on Volume Visualization (SIGGRAPH)*, pages 63–70, 1990. 1, 2
 14. Claudio T. Silva and Joseph S. B. Mitchell. The Lazy Sweep Ray Casting Algorithm for Rendering Irregular Grids. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):142–157, 1997. 1, 2
 15. Claudio T. Silva, Joseph S. B. Mitchell, and Peter L. Williams. An Exact Interactive Time Visibility Ordering Algorithm for Polyhedral Cell Complexes. In *Proceedings of the 1998 Symposium on Volume Visualization (VOLVIS-98)*, pages 87–94. ACM Press, 1998. 2, 6
 16. Clifford Stein, Barry Becker, and Nelson Max. Sorting and Hardware Assisted Rendering for Volume Visualization. In *Proc. 1994 Symposium on Volume Visualization*, pages 83–90. ACM SIGGRAPH, 1994. 1
 17. Rüdiger Westermann and Thomas Ertl. The VS-BUFFER: Visibility Ordering of Unstructured Volume Primitives by Polygon Drawing. In *Proc. IEEE Visualization 1997*, pages 35–42, 1997. 1
 18. J. Wilhelms and A. van Gelder. A Coherent Projection Approach for Direct Volume Rendering. *Computer Graphics*, 25(4):275–284, 1991. 1
 19. Peter L. Williams. Visibility Ordering Meshed Polyhedra. *ACM Transactions on Graphics*, 11(2):103–126, 1992. 2, 6
 20. Peter L. Williams and Nelson Max. A Volume Density Optical Model. *1992 Workshop on Volume Visualization*, pages 61–68, 1992. 2
 21. Peter L. Williams, Nelson L. Max, and Clifford M. Stein. A High Accuracy Volume Renderer for Unstructured Data. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):37–54, 1998. 2
 22. C. M. Wittenbrink. CellFast: Interactive Unstructured Volume Rendering. In *IEEE Visualization '99 Late Breaking Hot Topics*, pages 21–24, 1999. 6
 23. Roni Yagel, David M. Reed, Asish Law, Po-Wen Shih, and Naeem Shareef. Hardware Assisted Volume Rendering of Unstructured Grids by Incremental Slicing. In *Proc. IEEE 1996 Volume Visualization Symposium*, pages 55–62, 1996. 1
 24. Hansong Zhang. Vertex Program 1.1 and Texture Shader 3. *Game Developers Conference*, 2002. 3

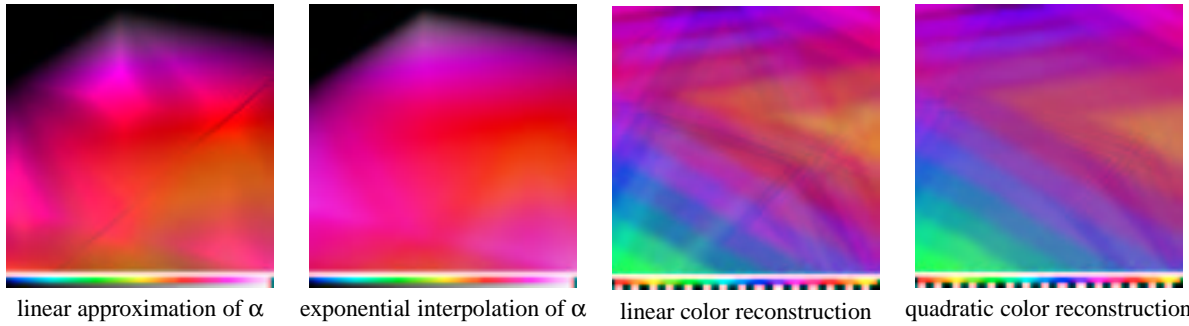


Figure 7: Comparison between different approximations of the three-dimensional ray integral. The applied transfer functions are depicted below each image.

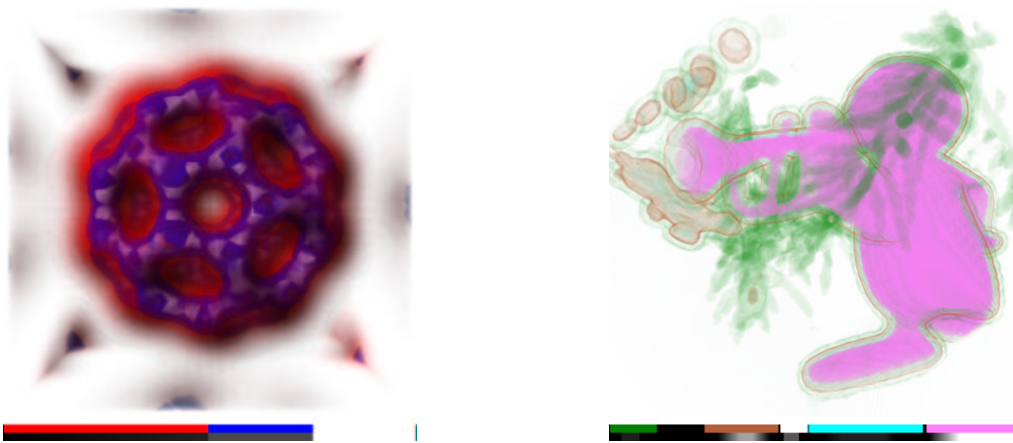


Figure 8: Bucky Ball with per-vertex lighting of original data set and part of the Christmas Tree data set⁴, both with quadratic polynomial approximation of chromaticity and accurate 16 bit α .

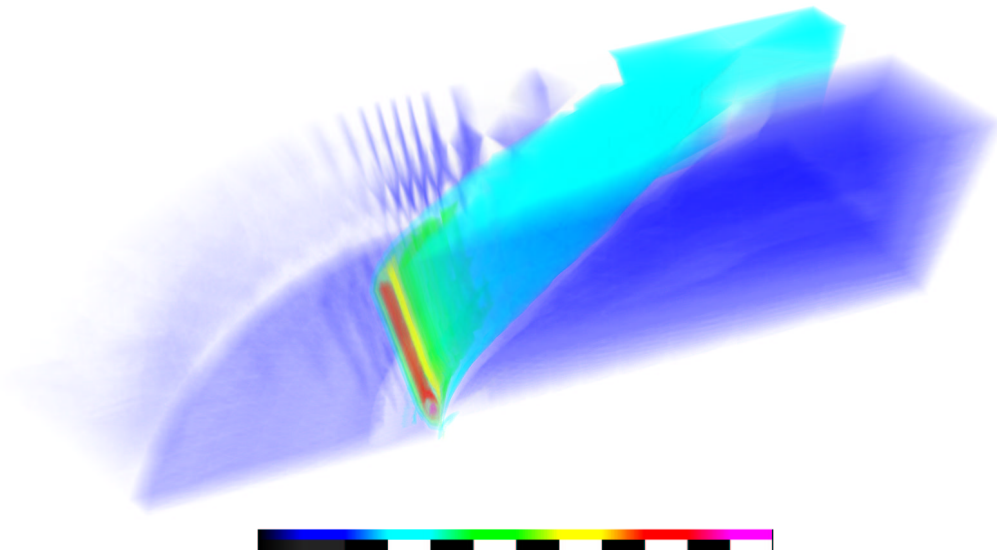


Figure 9: Blunt Fin dataset using quadratic polynomial approximation of chromaticity and 16 bit α . Due to the high reconstruction quality of the pre-integration table, the undersampling within the original data set can easily be seen.