Interactive Lighting Models and Pre-Integration for Volume Rendering on PC Graphics Accelerators

Michael Meißner
Viatronix Inc., Stony Brook, USA

Stefan Guthe
WSI/GRIS, Univeristy of Tübingen

Wolfgang Straßer
WSI/GRIS, Univeristy of Tübingen

Abstract

Shading and classification are among the most powerful and important techniques used in volume rendering. Unfortunately, for hardware accelerated volume rendering based on OpenGL, direct classification was previously only supported on SGI platforms and shading could only be approximated inaccurately, resulting in artifacts mostly visible in darkening.

In this paper, we present a novel approach for accurate shading of complex lighting models using multi-texturing, dependent textures (e.g. cube maps), and register combiners. Additionally, we present how different material properties can be integrated as a per voxel property to allow for more realistic image synthesis. Furthermore, we present a new technique circumventing the shading artifacts of previous approaches by pre-integrating an interpolation weight. Finally, we discuss how texture compression can be integrated to reduce the memory bandwidth required for relatively large volumes.

Key words: Volume Rendering, Texture Mapping Hardware, Multi-Texturing, Dependent Textures, Phong Shading, Classification, Pre-Integration.

1 Introduction

Due to the large amount of data, computations, and tremendous bandwidth requirements, software approaches are usually limited and far from interactive frame updates. One well known exception might be the ShearWarp algorithm [10], which can achieve interactivity taking advantage of optimizations such as run length encoding (pre-processing). However, each time the classification changes a new run length encoding needs to be calculated, and hence, for a fully occupied dataset with semi-transparent classification, no interactivity can be achieved on a desktop machine.

To overcome the inherent large amount of computation and the extreme bandwidth, texture mapping hardware has evolved to become the best known practical volume rendering method for rectilinear grid datasets. Despite the wide availability, texture mapping based volume rendering has some severe limitations: classification is a key technique in volume rendering interpreting the volume data as color, opacity, and material properties. To enable classification in texture mapping based volume rendering, a lookup is needed right after the texture mapping stage. Unfortunately, such a lookup was previously only available on a few platforms and is limited to the assignment of color and opacity but no further material properties can be integrated. Shading is yet another key technique to add further visual cues to the rendered images and enables a better interpretation of the images. In contrast to polygon rendering where a normal is a vertex property, a gradient is a voxel property. When using texture mapping for rendering volume data, no gradient estimation is supported in hardware. To circumvent this limitation, one can store the pre-calculated gradient together with the volume data as first proposed by Westermann et al. [19]. Despite the fact that many improved techniques have been proposed based on this approach, the subsequent shading operations of all of them [19, 15, 17] are based on not normalized interpolated gradients, resulting in shading artifacts and requiring that pre-normalized gradients are stored in the texture memory.

In this paper, a new approach for integrating lighting models into texture mapping based volume rendering on PC graphics hardware is presented. Furthermore, a new technique accomplishing the integration of classification for RGBA and material properties without the need of regenerating the entire texture nor requiring a second volumetric texture is described. Moreover, we present how the shading quality can be improved significantly when using pre-integrated classification, circumventing the artifacts of previous approaches.

1.1 Related Work

3D texture mapping hardware is recognized as a very efficient acceleration technique for volume rendering, since the first SGI RealityEngine [1] has been shipped. Cabral et al. [2] render datasets of 256^3 voxels at interactive frame-rates on a four Raster Manager SGI RealityEngine

Onyx with a single 150 MHz CPU. Similar results are presented by Cullip and Neumann [3]. The major drawback of the general texture mapping approach is the absence of shading functionality for volume data. To circumvent this, Van Gelder et al. [7] propose a 3-4 parameter lookup which is used to classify and shade the data. Unfortunately, no direct hardware support for such a lookup is available. Therefore, each time the viewing or classification changes, an entire new 3D texture needs to be generated.

Westermann et al. [19] store density values and corresponding pre-computed and pre-normalized gradients in texture memory and extensively exploit OpenGL and extensions for unshaded volume rendering and shaded isosurface rendering. Meißner et al. [15] extend this approach combining classification and diffuse shading for semi-transparent rendering of volume data. While both approaches use a matrix multiplication to obtain the diffuse shading intensity, Rezk-Salama et al. [17] use register combiners as available on the NVIDIA GeForce2. Despite of the impressive visual results, all these approaches [19, 15, 17] are based on not normalized interpolated gradients which result in shading artifacts, as explained later in this paper. Similarly to Westermann [19], Dachille propose to use the available hardware for efficient sample computation and possibly for blending [4]. Shading is performed on the host to ensure high quality rendering, thus avoiding the problem of non normalized gradients but interactivity is sacrificed.

Alternatively, the VolumePro board can be used to accomplish real-time frame-rates [16] but despite its superior performance, it offers less programmability than texture mapping and pre-integration is not feasible since dependent texturing is limited to a 1D lookup.

The remainder of this paper is organized as follows: Section 2 briefly summarizes the state-of-the-art in texture mapping based volume rendering, Our new artifact free shading approach for texture mapping based volume rendering is presented in Section 3. Section 4 presents a novel method for combining shading and pre-integrated classification, circumventing artifacts by pre-integrating an interpolation weight for the gradients. The necessary texture configuration is described in Section 5 and the results are summarized in Section 6. Finally, we conclude our paper and outline future work.

2 Texture Mapping Revisited

The shipment of the first SGI RealityEngine made 3D texture mapping hardware an available interactive feature. With respect to volume rendering, slicing planes parallel to the viewing plane are put through the volume in back to front order, see Figure 1(a). When using perspective projection, this becomes more complicated since one needs to account for the correct blending. Opacity values represent the volumetric absorption along a unit length and hence, one would need to use spherical shells [11] or either recompute the opacity values (see formula 4) or use dependent textures to correct the opacity which would suffer form the limited texture resolution (8 bit).

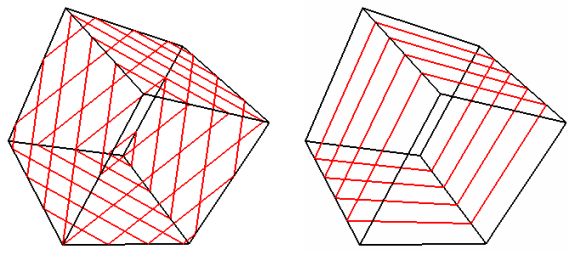


Figure 1: While in 3D texture mapping (a) arbitrary planes can be positioned in the volume, 2D texture mapping (b) requires a texture stack for each major viewing direction and the one most perpendicular to the actual viewing is selected.

Previously, the limited availability of 3D texture mapping prevented this approach from being widely used. Therefore, an alternative method — derived from the ShearWarp algorithm — has become popular. Three stacks of 2D textures are used, one for each major axis (see Figure 1(b)). Depending on the viewing vector, the stack most perpendicular to the viewing direction is used. To account for accurate volumetric absorption, opacity values need to be corrected depending on the viewing angle. Due to performance issues of 3D textures, this approach is still worth to be used when applicable.

2.1 Classification

Classification after resampling the volumetric data can be realized using post texturing lookup tables. Earlier, this could only be realized on mid- and high-end SGI platforms applying SGI's GL_TEXTURE_COLOR_TABLE_SGI. Exploiting multi-pass rendering, classification can also be accomplished using pixel textures, as presented in [15]. Unfortunately, pixel textures are not available on all platforms and due to the nature of multi-pass, the performance is reduced significantly. Another approach uses two volumetric textures and multi-texturing hardware, accomplishing classification in a single-pass [17]. Nevertheless, the approach requires two volumetric textures significantly increasing the memory requirements even if paletted textures are used. Furthermore, this approach can not be combined

with tri-linear interpolation based on two bilinear interpolations and register combiners. Other recent approaches make use of dependent texturing to accomplish classification [6, 9]. So far, only RGBA values are provided but no material properties.

2.2 Shading

As mentioned in the introduction, there has been a number of publications presenting shading of interpolated sample values within the context of texture mapping based volume rendering [19, 15, 17, 6]. Despite the fairly reasonable shading effects, all these approaches pre-compute the voxel gradient which is normalized, scaled, and biased in order to obtain gradient values of range $[0, 1]$. The gradient components are then stored in the RGB channels of an RGBA texture and the density value goes into the A channel. Using traditional texture mapping hardware, the gradient components and density value are interpolated and the scalar product is computed using using register combiners [17, 6, 9]. The severe drawback of all of these approaches is that they compute the scalar product using not normalized interpolated gradients. Thus, resulting in severe shading artifacts, mostly noticeable as darkening of the images. Furthermore, this also occurs for fairly smooth non binary datasets because the gradients at grid position need to be pre-normalized which again can introduce big differences of the gradient values of neighboring voxels, e.g. $(1, 0, 0)$ and $(0, 1, 0)$ enclose a 90 degree angle and in the worst case, the interpolated gradient will not be of length one but $\sqrt{0.5}$ causing the earlier mentioned darkening artifacts.

2.3 Gradient Magnitude Modulation

Using the gradient magnitude to suppress data which resides within homogeneous areas of a dataset is a very powerful feature for enhancing material boundaries [12]. Besides the magnitude of the first derivative, also the magnitude of the second derivative can be used to accomplish better visualizations of features within the dataset [8, 9].

Generally, when applying gradient magnitude modulation, the quality of the boundary enhancement depends mainly on the quality of the used gradient filter. While the intermediate and central difference gradient filters are prone to artifacts — since they result in non symmetric gradients —, the Sobel operator is the gradient operator of choice and used throughout this paper. Figure 7(d), (e), (f), and (k) show images using gradient magnitude modulation.

2.4 Pre-Integration

Pre-integrated classification is a technique used in volume rendering when classification is applied after interpolation [14, 6]. Following the Nyquist theorem, one can generally ensure that the reconstruction of the volumetric function along the rays is accurate. However, a non continuous transfer function, e.g. binary classification with infinite frequencies, introduces well-know slicing artifacts, as shown in Figure 2(a) and (c)).

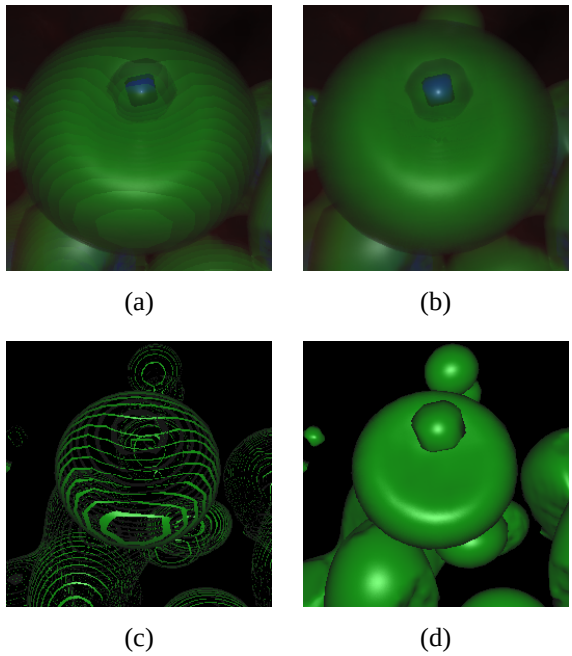


Figure 2: Pre-integrated volume rendering: Sample distance is 0.5 using plain ray casting without ((a) and (c)) and with pre-integration ((b) and (d)). The high frequencies of the iso-surface-like transfer function in (c) can be realized using pre-integrated sample intervals (d).

To circumvent this, pre-integration assumes a certain behavior of the volumetric function along the cast ray, e.g. linear. Based on the conventional 1D classification table, each interval between two samples can be pre-integrated and stored in a 2D table. During rendering, two consecutive sample values are used as indices for the 2D table, instead of classifying each individual sample assuming the color to be constant for the distance to the next sample along the ray. The advantage of pre-integrated volume rendering is that even precise iso-surfaces can be rendered without any additional cost during ray casting, see Figure 2(b) and (d). While Max et. al [14] pre-integrated opacity only, Engel et al.[6] extended this to RGBA but yet no material properties were considered. Generally, pre-integration suffers from shading artifacts when using more than one iso-surface or semi-transparent rendering, as explained in more detail in Section 4.

One of the main draw-backs of pre-integrated volume rendering is its incompatibility with gradient magnitude

modulation. Since the color is pre-integrated based on the voxel value only, no gradient magnitude modulation is possible, unless using a 4D function which would prevent interactivity. Solving this is still topic of research.

3 Lighting Models

Phong illumination is the most commonly used illumination model in volume rendering applications. The local illumination is split into three independent components and can be written as:

$$C^{\lambda} = k_a * \sum_{i=0}^{n} L_i^{\lambda} + k_s * \sum_{i=0}^{n} (\vec{N}\vec{R})^{n_s} * L_i^{\lambda} + k_d * \text{Class}^{\lambda}(v) * \sum_{i=0}^{n} *(\vec{N}\vec{L_i}), \quad (1)$$

where C^{λ} is the resulting color of wavelength λ, k_a, k_d, and k_s are the ambient, diffuse, and specular material properties, L_i is the color of lightsource i, v is the density value, Class$^{\lambda}$ is the classified color, $\vec{N}$ is the gradient, $\vec{R}$ is the reflected eye vector, $\vec{L}$ is the vector to the light-source, and n_s is the exponential factor to determine the size of the specular highlight.

Evaluating the Phong illumination requires normalized vectors in order to obtain the correct scalar products. However, when using pre-computed gradients stored in the volumetric texture, the length of the interpolated gradients is ≤ 1. The only approach to correctly solving this issue in the graphics pipeline is the use of cube-maps.

3.1 Cube-Maps

Generally, cube-maps are used to map the information contained in the scene onto the faces of a unit cube. During rendering, this information can be retrieved from the cube-map and projected onto the rendering primitive. One intuitive application of this are environment maps were the cube-map contains the projected RGB information of the scene, thus allowing glossy objects to reflect the environment without the need of sampling the real environment during rendering.

The same approach can be used to perform shading. Instead of projecting the colors of the objects of a scene onto the cube-map, the light sources can be projected onto the cube-faces and during rendering true shading for any number of light sources can be accomplished. This approach can also be extended to reflected components but requires the computation of the reflected vector [18]. Fortunately, the computation of the reflected vector recently became available in graphics accelerators since it is needed for bump- and environment-mapping. Thus, true Phong shading can be accomplished, e.g. on the NVIDIA GeForce3 or the ATI Radeon 8500.

For the application of shading in volume rendering, the interpolated gradient can be used to access a diffuse cube-map and compute the reflected vector to access a reflectance map containing the specular components [5]. Figure 3 (a) and (b) illustrate the diffuse and the specular cube-map for a single light-source. To support light

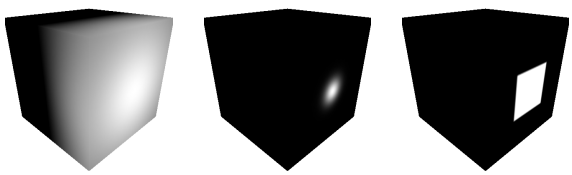


Figure 3: Cube-maps for one light source: (a) Diffuse cube-map. (b) Specular cube-map using a phong exponent of 50. (c) Specular cube-map of a squared light source.

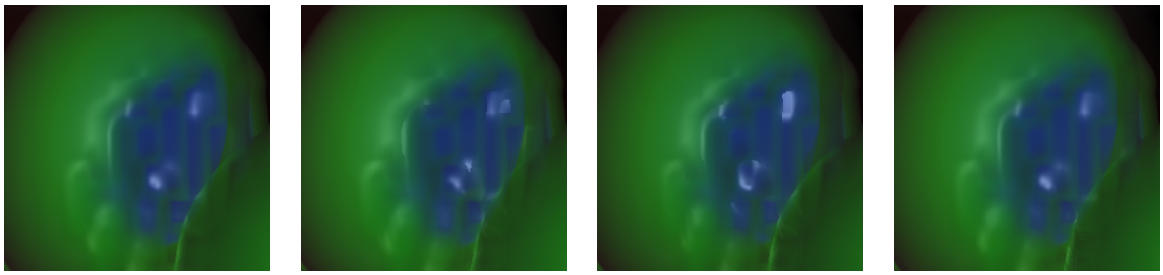
sources of different color, the specular cube-map needs to be of format RGB, resulting in images as shown in Figure 7(c). With the diffuse and specular light intensity, one can evaluate the Phong illumination using register combiners. The advantages of cube-maps are the available hardware support and the fact that they only need to be rebuilt when the light sources change the position relative to each other. For each light source it takes approx. 20 ms to build the corresponding cube-map which can easily be done interactively. Generally, the generation of the cube-maps is fill-rate limited but sizes larger than 64^2 do have a significant performance issue, presumably because they exceed what can be residing within the texture cache besides the 3D textures.

3.2 Different lightsources

Generally, any type of light source that can be represented in a cube-map can be realized. Thus, besides conventional point light sources also squared or arbitrary shapes of light sources are feasible. Our current implementation supports squared and rectangular shaped light sources as well as point light sources, possibly residing within the volume. The specular cube map for a squared light source is given in Figure 3(c) and the resulting rendering of the Neghip is shown in Figure 7(b).

4 Pre-Integration And Shading

Pre-integrated classification of ray intervals prevents artifacts due to the use of non continuous transfer functions. While pre-integration works well for opaque iso-surface or non-shaded volume rendering, it can not be combined correctly with semi-transparent volume rendering or when visualizing more than one iso-surface. This



(a) (b) (c) (d)

Figure 4: Pre-integrated classification and sample based shading in a software implementation. Blue material is opaque and highly specular while green material is semi-transparent and mainly diffuse: (a) Ray casting using 0.01 sample distance (b) Pre-integrated ray casting (sample distance 0.5) using the gradient and material properties of the first sample for shading. (c) Same as (b) but using the gradient and material properties of the second sample. (d) Ray casting using a pre-integrated weight to linearly interpolate the gradient used for shading.

is due to the fact that the color is pre-integrated for a given interval but the shading is performed on the fly at discrete ray sample positions. For any given voxel interval $[v_l, v_r]$, the two gradients at sample location are available but in most cases neither of the two allows for correct shading of this interval.

Engel et al.[6] approached this problem by correspondingly weighting the two gradients based on a global iso-surface value. Consequently, the offset of this iso-surface value v_i within the interval is computed during rendering and divided by the voxel interval length. The resulting weight

$$w = (v_i - v_l)/(v_r - v_l) \tag{2}$$

is used to perform a linear interpolation and the interpolated gradient is used to perform shading. While this approach correctly reveals the illumination for a single iso-surface v_i (see Figure 5(a)), it can fail when displaying more than one iso-surface (b) and generally fails for semi-transparent classification (c); e.g, in case of two iso-surfaces (v_i, v_j) being present, the method will fail for all intervals where the iso-surface v_i is behind the second iso-surface v_j which is orientation dependent. As a result, the gradient of the occluded iso-surface v_i is used for shading the visually contributing iso-surface v_j. The problem becomes worse for more than two iso-surfaces or for semi-transparent classification.

As a natural extension to pre-integration, we propose a new method solving this problem by pre-integrating an interpolation weight that matches the location of iso-surfaces within the interval. Thus, no global iso-value is required and for each interval always the gradient of the closest iso-value can be used for shading. The weight w is pre-integrated, weighting it with the remaining transparency.

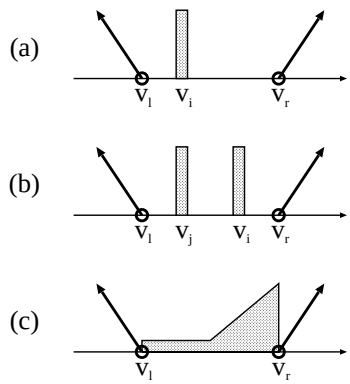


Figure 5: Shading of a pre-integrated color interval of a ray showing the sample location and the sample gradients. Left is voxel value v_l and right is v_r. (a) Single iso-surface being present. (b) Two iso-surfaces being present. (c) Semi-transparent classification along the interval.

$$w_{[v_l,v_r]} = \sum_{i=v_l}^{v_r} \frac{(v_i - v_l)}{(v_r - v_l)} \prod_{j=v_l}^{i-1} (1 - \alpha(v_j)) \tag{3}$$

To account for the correct remaining transparency in dependence of the interval length, the opacity $\alpha(v_j)$ needs to be corrected for each pre-integrated interval [13].

$$\alpha(v_j) = 1 - \sqrt[n]{1 - \alpha(v_j)} \tag{4}$$

where $n = \frac{d}{v_r - v_l}$ is the individual interval length and d being the actual sample distance along the ray.

Pre-integrating the gradient interpolation weight ensures that for any number of iso-surface being present

in any interval, the corresponding gradient of the closest iso-surface is used for shading. Furthermore, for semi-transparent classification, it allows to weight the two sample gradients to match best the opacity distribution within each interval.

Figure 4 summarizes the issues of pre-integrated color values and shading being performed on a discrete sample base. The images show a zoomed view of the Neghip dataset and were generated using $< k_a, k_d, k_s > = < 0.1, 0.6, 0.3 >$ for the green material and $< 0.1, 0.3, 0.6 >$ for the blue material. The reference image was generated using a sample distance of 0.01 and conventional point based classification, see Figure 4(a). For the other images, a sampling distance of 0.5 and a 2D pre-integrated classification table was used. Figure 4(b) shows the result of using the gradient at sample location v_l and (c) was generated using the gradient at v_r. In both cases, the obtained shading effects are wrong, especially the reflected highlights. In contrast, using the pre-integrated interpolation weight to obtain the gradient representing the interval best (see 4(d)), hardly any difference to (a) can be distinguished.

It should be mentioned that when using different material properties for each voxel, the k_d factor needs to be pre-integrated together with the diffuse color to account for correct shading. In addition, k_a and k_s need to be pre-integrated for the interval such that the specular and ambient term are correctly representing the entire interval.

$$k_{a,[v_l,v_r]} = \sum_{i=v_l}^{v_r} \frac{(v_i - v_l)}{(v_r - v_l)} \prod_{j=v_l}^{i-1} k_a(v_j) \qquad (5)$$

$$k_{s,[v_l,v_r]} = \sum_{i=v_l}^{v_r} \frac{(v_i - v_l)}{(v_r - v_l)} \prod_{j=v_l}^{i-1} k_s(v_j) \qquad (6)$$

In summary, weighting the gradient based on the opacity distribution across the pre-integrated interval allows to accomplish high image quality while performing the shading /computation only once per interval $[v_l, v_r]$.

5 Texture Setup

Setting up the texture and shading environments for either the NVIDIA GeForce3 or the ATI Radeon 8500 to achieve interactive frame rates is not a trivial task. Due to the limited hardware resources of the GeForce3, we first describe the implementation on the Radeon 8500 and subsequently describe the differences for the GeForce3.

For both, the gradient magnitude modulation and pre-integrated volume rendering, two 2D classification textures are used. One contains the diffuse color of the sample and the other the material properties and the opacity. Depending on the mode used, these classification textures are either addressed using the voxel and the norm of the gradient (gradient magnitude modulation mode) or by using two voxels (pre-integrated classification). Corresponding to the mode, the textures are initialized accordingly. In case additional lighting is enabled, two cube-maps are used as described earlier (see Section 3).

For gradient magnitude modulation, the 3D texture contains the voxel value, the norm of the gradient, and possibly the norm of the second order derivative to support higher dimensional transfer functions as recently presented in [9]. In order to support additional lighting, a second 3D texture is needed containing the pre-computed Sobel gradients since there is no texture format with more than four channels available. Without gradient magnitude, one GL_RGBA texture suffices.

For pre-integrated volume rendering, two instances of one volumetric texture containing the voxel and the gradient components are needed to provide the two sample values for the 2D lookup. Note that the volume is stored only once on the graphics card but two texture units use the same 3D texture.

For either of the two modes, pre-integration or gradient magnitude modulation in combination with illumination, a total of six texture operations is needed. Thus, on the ATI Radeon 8500 both approaches can be realized in a single pass due to its six independent texture units. With respect to programming the fragment shaders, the same program can be used for both modes since the setup for gradient magnitude modulation:

- 3D texture with voxel, gradient norm, etc.
- 3D texture with gradient
- 2D texture for material properties and opacity
- 2D texture for diffuse color
- Cube-map for diffuse light intensity
- Cube-map for specular light intensity

and the setup for pre-integrated volume rendering:

- 3D texture with voxel and gradient
- 3D texture with voxel and gradient
- 2D texture for (pre-integrated) material properties and opacity
- 2D texture for (pre-integrated) diffuse color
- Cube-map for diffuse light intensity
- Cube-map for specular light intensity

are very similar, using the same addressing scheme. In the first texture lookup the voxel and the gradient are extracted. The gradient is rotated into the coordinate system of the light sources and the texture coordinates for addressing the classification textures are set up. It is worth to be mentioned that since we do a per-pixel matrix multiplication to rotate the gradients into the coordinate system of the light source, we are also able to implement point light sources residing within the volume. In the second texture lookup, the voxel colors and the lighting intensities are obtained and combined in the final fragment shader step.

To achieve a higher lighting quality for pre-integrated rendering, as mentioned in Section 4, the gradient components need to be interpolated based on the pre-integrated interpolation weight. Due to the additional dependency, a second rendering pass is currently unavoidable but this might change with upcoming new graphics accelerators.

In contrast to the Radeon 8500, the implementation on the GeForce3 is either limited to lighting only or we have to use multiple rendering passes. The gradient magnitude modulation as done in a single pass on the Radeon 8500, has to be split into a total of four passes to allow for our general lighting model. This can be reduced to three passes when using a uniform color for either the diffuse or the specular cube map. However, the pre-integration approach without weighted gradients requires five passes while the additional interpolation of the gradient using the pre-integrated weight can not be implemented, due to the 8 bit quantization in the pipeline. While this appears to be very advantageous for the Radeon 8500, this might be quite different once the next generation graphics chips are released.

5.1 Texture Compression

One of the main drawbacks when using texture mapping hardware for volume rendering including shading is the need for storing an RGBA texture containing the pre-computed gradients. This is necessary because there is no support for extracting gradients directly from the density volume, as e.g. done in VolumePro [16]. Thus, a significant amount of texture memory is required to store the additional gradient information. For 8 bit voxel values and a 256^3 volume, the memory requirements are increased from 16 MBytes to 64 MBytes. Thus for a graphics card with 64 MBytes of memory (texture and framebuffer memory), and a volume that is much larger than the available texture memory, the volume needs to be partitioned into bricks which are transfered from main memory to the graphics card when needed. Even with an AGP bus, this significantly reduces the overall performance and real-time frame-rates are beyond feasibility.

Recently, the Architecture Review Board (*ARB*) of OpenGL released an extension for texture compressions named ARB_texture_compression and supported on many PC graphics cards (Voodoo5, Radeon, GeForce). The compression is based on the s3tc algorithm and accomplishes a constant compression rate of four by packing 4×4 texels into a compact bit stream. Thus, datasets which are much larger than the available texture memory of the graphics card can still be rendered at interactive frame-rates. However, image quality is potentially sacrificed due to the lossy compression algorithm and the missing adaption scheme for gradient compression. Figure 6 illustrates the difference in image quality for a full (a,b) and a close-up view (c,d) of the engine dataset. While the global information and structure is still present,

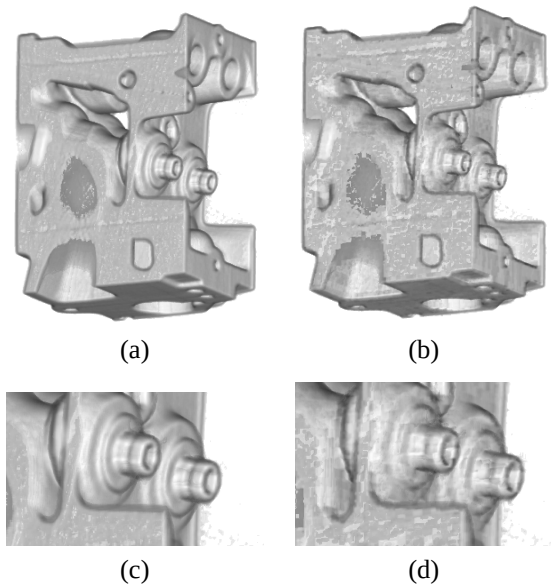


Figure 6: Texture compression applied to the engine dataset: (a,c) Without compression. (b,d) With compression.

fine detail is lost. Thus, one might want to implement a hybrid renderer, using compressed textures only during motion. Additionally, one can subdivide the volume into bricks and render from compressed textures only for the bricks being further away from the observer. However, such a hybrid rendering would require two different volumes which compete for residency on the graphics card.

6 Results

Several different datasets available on the volvis web page (http://www.volvis.org/) were used for rendering. The resulting images are shown in Figure 7 and clearly demonstrate the high quality of the presented techniques.

More interesting than the actual quality is the timing analysis. While all renderings can be accomplished on the NVIDIA GeForce3 and on the ATI Radeon 8500, the performance shows significant differences. This is due to the different hardware resources available on these systems. While the GeForce3 has four texture units, the Radeon 8500 provides six and has less restrictions when using cube-maps. E.g. when using 3D textures, shading, and classification, three passes are needed to accomplish the rendering on the GeForce3 while the Radeon 8500 is capable of handling this in one pass. While this is likely to constantly change with new upcoming graphics accelerators, the following results were measured on the ATI Radeon 8500.

We investigated several issues and their impact onto the overall rendering speed. First, the size of the viewport and the dataset determine the amount of tri-linear samples that need to be generated. Second, enabling or disabling shading because there are more textures to be used per polygon. Third, the size of the cube-maps which is a trade-off between quality and cache efficiency and last but not least, the impact of compression. For all measurements, the slice distance was chosen to be one and the size of the diffuse and specular cube-map were 16^2 and 64^2 respectively. Increasing the diffuse cube-map to 64^2 reduced the performance by 10% without further increasing the image quality.

Table 1 illustrates the timing using a viewport of 200×200 pixels. Simply slicing the density volume and applying a dependent 2D texture for classification is in the range of 3.6 to 82.9 frames. When slicing a density

Data size	no-light	cube-map	compression
64^3	82.8	36.1	40.1
128^3	18.4	12.8	19.6
$256^2 \times 128$	8.2	4.6	8.4
256^3	3.6	-	4.7

Table 1: Frames per second for a 200×200 *viewport.*

and gradient texture and applying the cube-maps including classification, the frame rate drops strongly. This is mainly due to the utilization of two volumetric textures per sample for pre-integration (halfing the performance) as well as applying the cube-map which competes for texture cache. In case of *no-light*, we are using the same RGBA texture containing the gradients and the performance can be improved by a factor of two using a GL_ALPHA texture. With the currently released drivers we were not able of getting cube-maps to run with a 256^3 dataset but this is likely to change and could be circumvented using bricking.

Enabling ARB_TEXTURE_COMPRESSION increases the rendering performance but not so much for the small datasets than for the larger datasets. The reason for this is that for larger datasets the memory bandwidth of the graphics card is the limitation and due to the compression, the cache efficiency is increased. However, for smaller datasets, the cache efficiency is already high and thus, texture compression can not significantly accelerate the rendering because the actual limitation here is the cube-map lookup.

Finally, table 2 shows the same timings for a viewport of 400×400 pixels. In comparison to Table 1, all timings are reduced by a factor of 2 to 2.9 due to the four times enlarged viewport. The factor reveals the increase in texture cache efficiency allowing a higher pixel fill-rate.

Data size	no-light	cube-map	compression
64^3	37.3	12.7	12.4
128^3	11.4	5.5	6.4
$256^2 \times 128$	5.0	2.5	3.9
256^3	2.3	-	2.5

Table 2: Frames per second for a 400×400 *viewport.*

On the Radeon 8500 3D textures are a factor of two to three slower than 2D textures and their performance varies depending on the viewing direction (linear memory access problems). However, true 3D textures are mandatory for pre-integrated classification because the pre-integration is based on a fixed interval length. Thus, when using 2D textures the interval length depends on the viewing direction and would need to be recomputed for every frame which is not possible at interactive framerates unless only iso-surface rendering is of interest.

7 Conclusions

In this paper, we presented a novel approach for accomplishing artifact free shading of volumetric data using cube-maps. This approach allows to not only support directional light sources but also any complex lighting situation. Additionally, the presented approach allows to specify material properties on a per sample base.

Furthermore, we presented how pre-integrated classification can be combined with shading by additionally pre-integrating an interpolation weight used to interpolate the two respective gradients at sample location.

The presented results were generated on a ATI Radeon 8500 using OpenGL but most of them could also be accomplished on a NVIDIA GeForce3 using multi-pass rendering due to the limited hardware resources. Besides its high throughput, the Radeon 8500 offers highest possible flexibility within the texturing and the rasterization

stage. As demonstrated, this flexibility can be efficiently exploited to enable and combine the most important and valuable techniques of volume rendering at interactive frame rates.

A topic of future work is to investigate the impact of the limited frame-buffer accuracy onto the image quality as well as investigating how pre-integrated classification and gradient magnitude modulation can be combined. Using a 4D table (v_s, v_e, gm_0, gm_1) would allow to handle this correctly but the table could not be computed interactively and other solutions are necessary.

References

[1] K. Akeley. RealityEngine Graphics. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 109–116, August 1993.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. In *Workshop on Volume Visualization*, pages 91–98, Washington, DC, USA, October 1994.

[3] T. J. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Mapping Hardware. Technical Report TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993.

[4] F. Dachille, K. Kreeger, B. Chen, I. Bitter, and A. Kaufman. High-Quality Volume Rendering Using Texture Mapping Hardware. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 69–76, Lisboa, Portugal, August 1998.

[5] S. Dominé and J. Spitzer. OpenGL Texture Shaders. *Technical document, available from http://www.nvidia.com/*, 2001.

[6] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, Los Angeles, CA, USA, August 2001.

[7] A. Van Gelder and K. Kim. Direct Volume Rendering With Shading via Three-Dimensional Textures. In *Symposium on Volume Visualization*, pages 23–30, San Francisco, CA, USA, October 1996.

[8] G. Kindlmann and J. W. Durkin. Semi-automatic generation of transfer functions for direct volume rendering. In *Symposium on Volume Visualization*, pages 79–86, Research Triangle Park, NC, USA, October 1998.

[9] J. Kniss, G. Kindlmann, and C. Hansen. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *Proc. of IEEE Visualization*, pages 255–262, San Diego, CA, USA, October 2001. IEEE Computer Society Press.

[10] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 451–457, July 1994.

[11] E. LaMar, B. Hamann, and K. Joy. Multiresolution Techhniques for Interactive Hardware Texturing-based Volume Visualization. In *Proc. of IEEE Visualization*, pages 355–361, San Franisco, CA, USA, October 1999. IEEE Computer Society Press.

[12] M. Levoy. Display of surfaces from volume data. *Ph.D. Dissertation, Department of Computer Science, The University of North Carolina at Chapel Hill*, May 1989.

[13] B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction to volume rendering*. Hewlett-Packard Professional Books, Prentice-Hall, Los Angeles, USA, 1998.

[14] N. Max, P. Hanrahan, and R. Crawfis. Area and volume coherence for efficient visualization of 3d scalar functions. pages 27–33, San Diego, CA, USA, nov 1990.

[15] M. Meißner, U. Hoffmann, and W. Straßer. Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions. In *Proc. of IEEE Visualization*, pages 207–214, San Franisco, CA, USA, October 1999. IEEE Computer Society Press.

[16] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro Real-Time Ray-Casting System. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 251–260, Los Angeles, CA, USA, 1999.

[17] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, and T. Ertl. Interactive volume rendering on standard pc graphics hardware using multi-texturing and multi-stage rasterization. In *Proc. of Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 109–118, Interlaken, Switzerland, August 2000.

[18] D. Voorhies and J. Foran. State of the art in data visualization. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 163–166, July 1994.

[19] R. Westermann and T. Ertl. Efficiently Using Graphics Hardware in Volume Rendering Applications. In *Computer Graphics*, Proc. of ACM SIGGRAPH, pages 169–177, Orlando, FL, USA, August 1998.

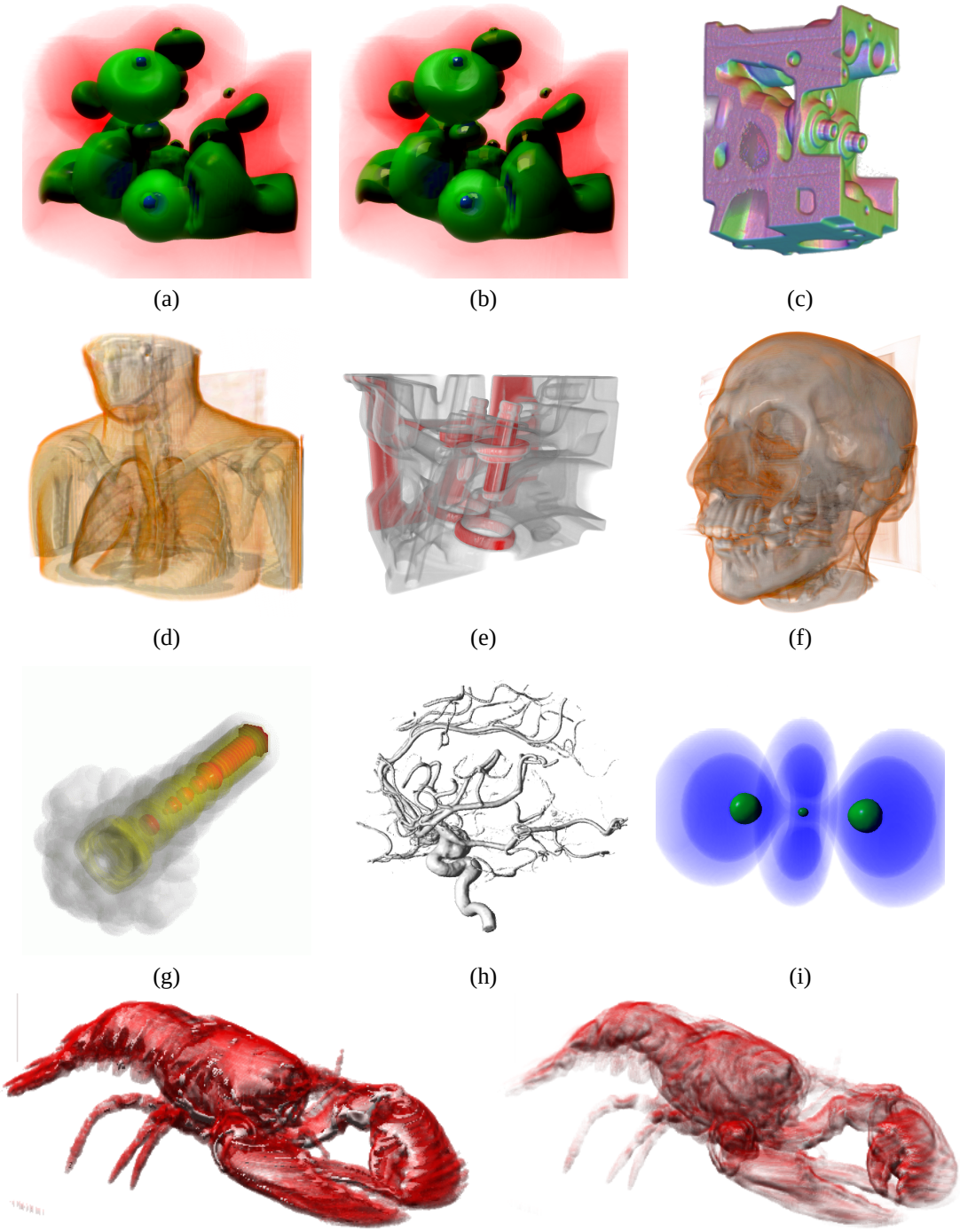


Figure 7: Color plates: gm *denotes gradient magnitude modulation and* mp *denotes different material properties. (a) Neghip using a point light source and mp. (b) As (a) using a squared light source. (c) Engine using differently colored light sources. (d) Visible human using gm. (e) Engine using gm and mp. (f) Head using gm. (g) Fuel using mp. (h) Aneurysm. (i) Hydro using mp. (j) Lobster. (k) as (j) using gm.*