Using pre-integrated transfer functions in an interactive software system for volume rendering

G. Knittel

WSI/GRIS, University of Tuebingen, Germany, knittel@gris.uni-tuebingen.de

Abstract

We explore possibilities to implement classification based on pre-integrated transfer functions in a software system for high-speed volume rendering. The major obstacle to remove is the large size of the multi-dimensional look-up tables. For high rendering performance, all parameters must be available in the L1-cache of the CPU, which is usually by far too small to hold the required tables. We discuss a cache-efficient and fast approximate solution and present a comparison to the standard classification method with respect to performance and image quality.

1. Introduction

The use of pre-integrated transfer functions was introduced by Max et al. [5] and Engel et al [3]. In [3], the authors give an approximation to the opacity α and the emitted light intensity $\tilde{C}$ of an interval between two raypoints having a distance d and scalar values of s_b and s_f, respectively, as

$$\alpha(s_f, s_b, d) \approx 1 - \exp\left(\frac{d}{s_b - s_f} \cdot (T(s_f) - T(s_b))\right) \tag{1}$$

and

$$\tilde{C}(s_f, s_b, d) \approx \frac{d}{s_b - s_f} \cdot (K(s_b) - K(s_f)) \tag{2}$$

with the integral functions

$$T(s) = \int_0^s \tau(s)\mathrm{d}s \qquad \text{and} \qquad K(s) = \int_0^s \tilde{c}(s)\mathrm{d}s \tag{3}$$

In (3), $\tau(s)$ is the extinction density. Color density $\tilde{c}(s)$ has often the form $c(s) \cdot \tau(s)$, c being r, g or b. Assuming d as constant, the authors compute a two-dimensional table from the user-supplied transfer functions $\tau(s)$ and $c(s)$, which is stored in the texture memory of a suitable graphics accelerator. Rendering is done using texturing hardware. Classification is performed by accessing the table using the values of the actual and the previous raypoint. The method can reduce aliasing introduced by high frequencies in the user-supplied transfer functions. Note that this is the simplified version, which does not consider self-attenuation within a ray interval. It was proposed by the authors to reduce computation times for the table. This is also the basis for our software implementation, although for different reasons.

For an efficient software implementation any large look-up table should be avoided, even at the costs of more complex computations. This is because modern CPUs are bandwidth-limited, and perform well only if all data items are available in the on-chip cache (L1-cache). The two-dimensional table addressed by s_b and s_f is too large to fit in an L1-cache of usual size. Assuming 8 bits for both addresses and 64 bits per entry, the table would occupy 512Kbytes. Furthermore, there is no locality in the access pattern. Thus, using a table like this will result in a high number of cache misses. The resulting memory reads, however, are the most severe performance drain. Literally hundreds of CPU-cycles can be wasted during a single cache line fill.

In the following we will discuss a cache-efficient implementation which uses only a one-dimensional table containing the integral functions. Equations (1) and (2) are computed on-the-fly using advanced features of modern CPUs. Rendering performance and image quality is compared to the standard classification at the end of the paper.

2. Implementation

2.1. Processor features

The raycasting routines have been implemented in assembly language for the Intel Pentium-4 CPU. They make use of the MMX- and SSE-units for SIMD-computations. For a comprehensive description of these techniques, please refer to [1]. Here we will explain only the necessary details. The SSE-unit provides eight registers, each 128 bits wide. They can be loaded with four single-precision floating-point operands. An SSE-instruction can process four such operands or operand pairs in parallel. Instructions referred to in this paper compute an approximation to the reciprocals of the operands contained in one register (RCPPS), and the products, sums or differences of the operands in a register pair (MULPS, ADDPS and SUBPS), respectively. The MINPS-instruction returns the minima of four operand pairs. ADDPS, SUBPS and MINPS have a latency of four, MULPS and RCPPS a latency of six cycles [2].

Also mentioned must be the conditional move instruction CMOVcc, which is either skipped or moves the contents of one integer register into another, depending on the condition code *cc* set by a preceding logical, arithmetic or compare operation. This operation can be used to avoid conditional branches.

2.2. Arithmetic evaluation

As first approximation we assume that the raypoint distance *d* is constant for all rays, and can therefore be multiplied into the integral functions. The integral functions as given in equation (3) are pre-computed as one summed table with 256 entries. Each entry contains the accumulated associated colors $K(s) = \tilde{R}_A, \tilde{G}_A, \tilde{B}_A$ and the accumulated opacity $T(s)$ as 32-bit floating-point numbers. Thus, the table size is only 4Kbytes and will result in a high hit ratio in the L1-cache.

During rendering, the raycaster operates such that always the newly interpolated value at the current raypoint location as well as the previous raypoint value are available. The larger and smaller of the two values, here called s_{max} and s_{min}, are determined using CMOVcc instructions. The difference is converted into floating-point format while being transferred into an SSE-register. The RCPPS-operation creates the (positive) denominator. Using s_{max} and s_{min}, two sequential table accesses are performed. A SUBPS followed by a MULPS delivers the (positive) raypoint light emission $\tilde{C}(s_{\max}, s_{\min})$.

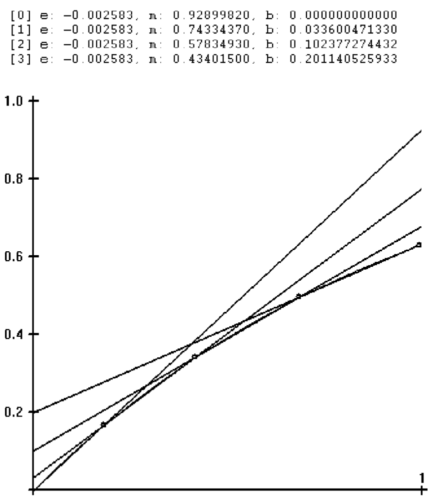
The remaining problem is the evaluation of

$$1 - \exp(-x)\text{, where } x = \frac{T(s_{\max}) - T(s_{\min})}{s_{\max} - s_{\min}} \geq 0 \qquad (4)$$

This must be done in only a few clock cycles. The basic idea is to approximate this function by four lines $m_i \cdot x + b_i$, and to evaluate the four expressions by one MULPS- and one ADDPS-instruction for the given *x*. The correct value is the minimum of the four results, which can be found using two MINPS-operations (data movement and format conversions are not detailed here).

Assuming $0 \leq \tau < 1$ it follows that $0 \leq x < 1$ and so the parameters m_i and b_i can be computed once for all times by any brute-force method. The approximation error is sufficiently small, as shown in Figure 1. For reader's interest, an approximation for $0 \leq x \leq \infty$ has a maximum absolute error of 0.022593, about nine times as much, but is probably still useful for many applications.

In summary, there are three memory read accesses necessary: one for the previous raypoint value, and two for the table entries, for a total of 33 bytes. Using an optimized memory layout, it has been shown that most of these read requests can be satisfied out of the L1-cache [4]. Memory writes can in



```
[0] e: -0.002583, m: 0.92899820, b: 0.000000000000
[1] e: -0.002583, m: 0.74334370, b: 0.033600471330
[2] e: -0.002583, m: 0.57834930, b: 0.102377274432
[3] e: -0.002583, m: 0.43401500, b: 0.201140525933
```

Figure 1: *Approximation of $1\text{-}e^{-x}$*

most cases be hidden. Most other computational expenses are fast arithmetic vector operations, so that a high performance can still be expected.

2.3. Thresholds and empty-space-skipping

Often, small values in the data set are noise and should not contribute to the image. Such raypoints can be discarded by setting the transfer functions correspondingly. In software systems it can be faster to compare the raypoint value directly after tri-linear interpolation to a user-supplied threshold value, and to skip the otherwise useless table accesses and compositing operations. However, a raypoint below the threshold may not simply be discarded, since the (linear) function between this and the previous ray-point may exceed the threshold in a fraction of the interval. Provided the transfer functions are clamped to zero below the threshold as well, processing of the raypoint in question will still deliver the intended result.

However, a raypoint *can* be discarded if s_{max} is below the threshold (which was the primary reason to sort the two values). This is shown in Figure 2. Depending on the data set and rendering parameters, a significant speed-up can still be achieved.

Empty-space-skipping is basically the same, although on a larger scale and often with the use of a hierarchical data base. For entering and leaving empty space, however, the same principles apply.

3. Results

The methods were integrated in the UltraVis [4] system. The test data set was the “neghip” data set, with

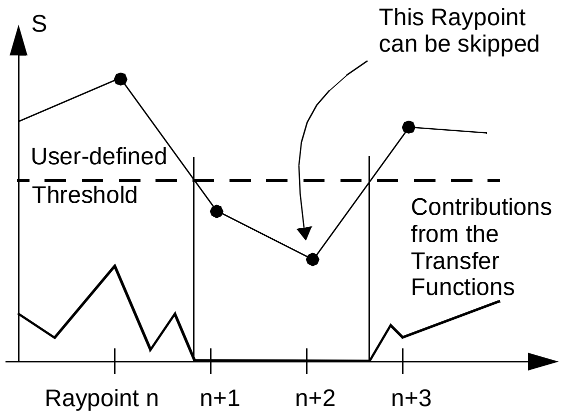

S
This Raypoint can be skipped
User-defined Threshold
Contributions from the Transfer Functions
Raypoint n
n+1
n+2
n+3

Figure 2: *Using threshold values.*

a resolution of 64×64×64 voxels. The voxels have been placed in memory redundantly (i.e., each voxel was present eight times), so that all voxels for tri-linear interpolation could be obtained from memory in a single access. Early-ray-termination was applied when the translucency fell below 0.004. Empty-space-skipping (ESS) was used if it provided a speed-up. A threshold was set to 32. A total of 256×256 rays were shot to produce images of the same size. The (somewhat arbitrary) transfer function is shown in Figure 3. The test machine was a Pentium-4 PC running at 2GHz with 1Gbyte RDRAM (RAMBUS) main memory.

The images are shown in Figure 4a through Figure 4f. On the left side, the raypoint distance d was set to 0.45 grid units, on the right side to 0.9. The top row was generated with standard classification (point sampling in the transfer function tables). The middle

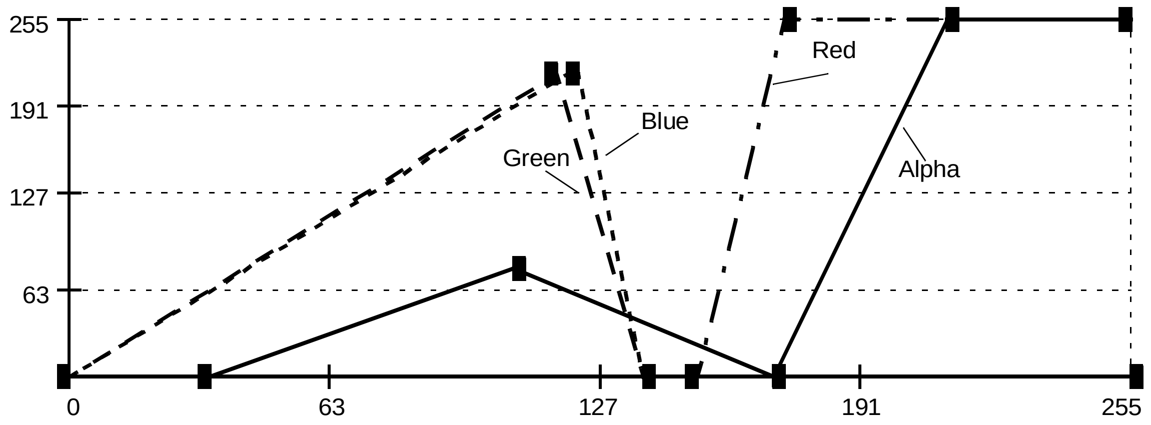


Figure 3: *Transfer Functions*

row was rendered with pre-integrated transfer functions as described in section 2.2. For the bottom row, however, the opacity α was computed exactly like the color components as given in equation (2), i.e., without the approximation of $1 - \exp(-x)$.

As found by the inventors, the proposed methods can reduce aliasing artifacts for a given raypoint distance substantially. The question for an interactive software implementation is of course whether or not a speed-up can be achieved. This is indeed the case: using Figure 4a (standard classification) as a reference, it can be seen that even by doubling the raypoint distance a comparable image quality can be obtained. Then the proposed methods provide the highest frame rates. Using the simplified α is probably the best compromise between speed and image quality, although a certain loss in contrast is clearly visible (cf. Figure 4d and Figure 4f).

The results are summarized in Table 1.

Image	d	Method	Rendering Time [s] Frame Rate [Hz]	ESS
a	0.45	Standard Class.	0.182 5.5	no
b	0.9	Standard Class.	0.082 12.2	no
c	0.45	Pre-Int.	0.295 3.4	yes
d	0.9	Pre-Int.	0.158 6.33	no
e	0.45	Pre-Int. simpl. α	0.253 3.95	yes
f	0.9	Pre-Int., simpl. α	0.132 7.6	no

Table 1: *Rendering results*

4. Conclusion and future work

This snapshot of ongoing work leads to the following conclusions. The software implementation was in so far successful as it achieved a speed-up for comparable image quality despite significantly increased computing expenses.

Alternative implementations, such as evaluation of $1 - \exp(-x)$ by table look-up, have been found to be slower. Also, reducing the table entries from 32-bit floating-point to 16-bit fixed-point operands didn't have a measurable effect on rendering speed.

Future work is dedicated to more fundamental research. Although the benefits of the use of pre-integrated transfer functions are undisputed, the method is also limited in several ways. First, it is no remedy against resampling artifacts resulting from high frequencies already present in the data set itself (cf. Figure 4b and Figure 4d). Second, it is limited in practice to simple scalar fields, post-classification and no or very simple shading models. A software system may be able, for example, to render pre-segmented data sets with gradients and apply diffuse and specular shading from multiple light sources using material-dependent rendering parameters. It is unclear how pre-integrated tables can be of practical use here.

Thus, other more general techniques must be examined and compared. An obvious candidate is rendering with an adaptive ray sampling rate, which constitutes our short-term research activity.

Acknowledgments

This work was done for the SFB 382 funded by the DFG, Deutsche Forschungsgemeinschaft. UltraVis is a joint Research Project of HP Labs, Palo Alto, CA, and WSI/GRIS. I would like to thank my colleagues Stefan Guthe and Andreas Schilling for many fruitful discussions.

References

1. **Anonymous,** *"IA-32 Intel Architecture Software Developer's Manual, Vol. 2"*, Intel Corp., Order Number 245471. 2
2. **Anonymous,** *"Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual"*, Intel Corp., Order Number 248966-04. 2
3. **K. Engel, M. Kraus, T. Ertl,** *"High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading"*, Proceedings Graphics Hardware 2001, ACM Press, pp. 9-16 (August 2001). 1
4. **G. Knittel,** *"The UltraVis System"*, Proceedings of the Volume Visualization and Graphics Symposium 2000, ACM Press, pp. 71-79 (October 2000). 3
5. **N. Max, P. Hanrahan, R. Crawfis,** *"Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions"*, in *Computer Graphics*, Proceedings of San Diego Workshop on Volume Visualization, pp. 27-33 (1990). 1

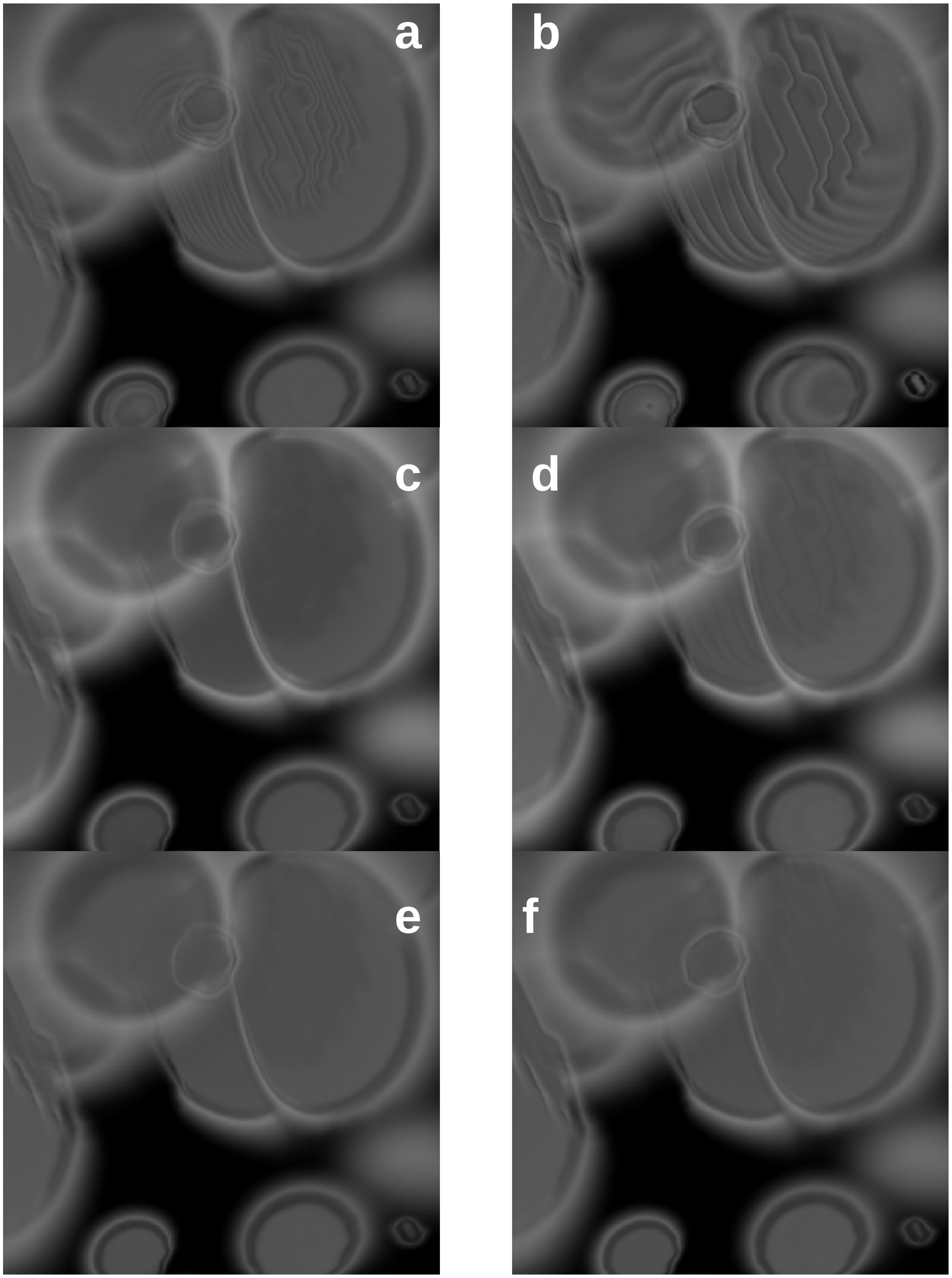


Figure 4: *Sample Images*