Volume Ray Casting with Peak Finding and Differential Sampling

Aaron Knoll *(Member, IEEE)*, Younis Hijazi, Rolf Westerteiger, Mathias Schott
Charles Hansen *(Senior Member, IEEE)* and Hans Hagen *(Senior Member, IEEE)*

Abstract— Direct volume rendering and isosurfacing are ubiquitous rendering techniques in scientific visualization, commonly employed in imaging 3D data from simulation and scan sources. Conventionally, these methods have been treated as separate modalities, necessitating different sampling strategies and rendering algorithms. In reality, an isosurface is a special case of a transfer function, namely a Dirac impulse at a given isovalue. However, artifact-free rendering of discrete isosurfaces in a volume rendering framework is an elusive goal, requiring either infinite sampling or smoothing of the transfer function. While preintegration approaches solve the most obvious deficiencies in handling sharp transfer functions, artifacts can still result, limiting classification. In this paper, we introduce a method for rendering such features by explicitly solving for isovalues within the volume rendering integral. In addition, we present a sampling strategy inspired by ray differentials that automatically matches the frequency of the image plane, resulting in fewer artifacts near the eye and better overall performance. These techniques exhibit clear advantages over standard uniform ray casting with and without preintegration, and allow for high-quality interactive volume rendering with sharp C^0 transfer functions.

Index Terms—direct volume rendering, isosurface, ray casting, ray differentials, sampling, transfer function, preintegration, view dependent

1 Introduction

Volume rendering is an indispensable tool in visualization, with applications ranging from simulation data analysis to imaging medical and biological scan data. The principal means of visualizing a volume consist of choosing an isosurface or interpreting the volume in its entirety via direct volume rendering (DVR). Traditionally, these modalities have been implemented using separate algorithms, and were employed with different visualization and application goals. Isosurfaces are commonly generated by extracting a triangle mesh, and are useful in understanding topological and geometric behavior of a scalar field at implicitly defined boundaries. Direct volume rendering is a more expressive method of visualizing volume data, in which the user supplies a transfer function mapping scalar values to colors. As opposed to a single isovalue corresponding to a 2-manifold surface, a transfer function allows for rendering 3-manifold segments of the volume.

In principle, an isosurface can be defined by a transfer function with a Dirac impulse at the chosen isovalue. However, rendering such a transfer function poses problems for most conventional volume rendering algorithms. Methods involving uniform sampling and postclassification will invariably miss an infinitely fine impulse, entirely omitting the desired surface features. Preintegrated transfer functions remedy this, but introduce new artifacts due to their discretization of scalar values into a 2D lookup table, and weighting assumptions on the volume rendering integral. Most commonly, the solution to rendering isosurfaces within a volume rendering framework has been to increase the sampling rate and to smooth the transfer function. Nonetheless, doing so can be computationally wasteful and limits classification.

Spatial traversal strategies for GPU isosurface ray casting closely mirror those for DVR sampling. We pair these processes by identifying isovalues of interest at peaks of a 1D transfer function, using the uniform volume ray casting process to isolate these roots, and sampling directly at the desired isovalues. While numerous applications allow for multi-modal DVR and isosurface visualization, to the best of our knowledge our approach of sampling isosurfaces directly within the volume rendering integral has not previously been employed. Perhaps this is because standard techniques employing smooth transfer functions were considered sufficient. Nonetheless, definition and accurate rendering of sharp transfer functions is desirable, not only in terms of overall image quality but in the ability to classify features flexibly and render accurately with a fixed sampling budget. To further ensure samples are spent wisely, we devise a novel approach to volumetric sampling using a quadratic function for incrementing samples based on ray differential propagation. This helps in sufficiently sampling features close to the viewpoint, and is particularly useful when employing higher-order filters for which samples are expensive. While orthogonal, these techniques work well together, particlarly in rendering nearby high-frequency features with high fidelity. We compare both methods with standard techniques, and show how they offer higher quality imaging and better classification for various data sets.

- *Aaron Knoll is with the University of Kaiserslautern, E-mail: knolla@rhrk.uni-kl.de*
- *Younis Hijazi is with LSIIT at the University of Strasbourg. E-mail: hijazi@lsiit.u-strasbg.fr*
- *Rolf Westerteiger is with the University of Kaiserslautern. E-mail: rolf.westerteiger@googlemail.com*
- *Mathias Schott is with the SCI Institute, University of Utah. E-mail: mschott@cs.utah.edu*
- *Charles Hansen is with the SCI Institute, University of Utah. E-mail: hansen@cs.utah.edu*
- *Hans Hagen is with the University of Kaiserslautern. E-mail: hangen@informatik.uni-kl.de*

Manuscript received 31 March 2009; accepted 27 July 2009; posted online 11 October 2009; mailed on 5 October 2009.
For information on obtaining reprints of this article, please send email to: tvcg@computer.org .

2 Related Work

Levoy [16] employed ray casting in the first implementation of direct volume rendering. The advent of z-buffer hardware and built-in texture interpolation units allowed for interactive performance with slice-based rasterization approaches [2, 4]. Similarly, rasterization methods employing splatting [32] proved to be efficient, particularly for applications involving unstructured data and higher-order reconstruction filters [34, 35]. While optimized CPU algorithms are capable of interactive volume rendering [11, 15], GPU approaches gained popularity, due to improved computational throughput and built-in texture fetching and interpolation. With programmable shader support for branching and looping, volume ray casting methods experienced resurgence on the GPU [14, 26].

The conventional means of rendering discrete isosurfaces from volume data has been to extract a mesh using marching cubes [20]. Mesh extraction methods can be combined with min-max spatial subdivision structures [33], as well as view dependent [18] approaches for further efficiency. Marching cubes only approximates the implicit surface on a coarse scale, and more sophisticated methods [28] are generally not suited for dynamic extraction. However, it is possible to combine extraction with splatting [19] for efficient rendering, or to employ splatting directly on isosurfaces [3].

Ray casting methods were first applied towards volumetric isosurfacing by Sramek [31]. Parker et al. [24,25] implemented a tile-based parallel ray tracer and achieved interactive rendering of isosurfaces from large structured volumes, employing a hierarchical grid as a min-max acceleration structure and an analytical cubic root solving technique for trilinear patches. Hadwiger et al. [7] combined rasterization of min-max blocks with adaptive sampling and a secant method solver to ray cast discrete isosurfaces on the GPU. Our peak finding method is close in spirit to this approach; however we employ our solving

method not only in rendering isosurfaces but in handling potentially sharp unidimensional transfer functions. Ray differentials were introduced by Igehy [8] as a way of calculating image-space derivatives of pixels as rays are transmitted, reflected and refracted in world-space, and using these values for filtering. While similar concepts have been used in multiresolution isosurface ray casting [13], to our knowledge no approach has used ray differentials for volumetric sampling.

A large body of volume rendering literature deals with transfer functions, both in how to construct them and employ them in classification. To limit artifacts when sampling high-frequency features of a transfer function, the best existing approaches are preintegration [5,21,27] and analytical integration of specially constructed transfer functions [10]. Hadwiger et al. [6] analyze the transfer function for discontinuities to generate a pre-compressed visibility function employed in volumetric shadow mapping. Our approach is similar except that we search for local maxima, and use these directly in enhancing classification.

3 Background and Overview

Direct volume rendering is the process of modeling a volume as a participating optical medium, and estimating the emission and absorption of these media according to a discrete approximation of the radiative transport equation. On a segment of a ray, irradiance is formulated as

$$I(a,b) = \int_a^b \rho_E(f(s))\rho_\alpha(f(s))e^{-\int_a^s \rho_\alpha(f(t))dt}ds \tag{1}$$

where ρ_E is the emissive (color) term and ρ_α is the opacity term of the transfer function; a,b are the segment endpoints, and $f(t) = f(\vec{O}+t\vec{D}) = f(\vec{R}(t))$ is the scalar field function evaluated at a distance t along the ray. To compute this integral, we must approximate it discretely. The conventional approach of Levoy [16] is to break up the ray into equally spaced segments, approximating the opacity integral as a Riemann Sum,

$$e^{-\int_a^s \rho_\alpha(f(t))dt} = \prod_{i=0}^{n} e^{-\Delta t\ \rho_\alpha(f(i\,\Delta t))} = \prod_{i=0}^{n}(1-\alpha_i) \tag{2}$$

where Δt is the uniform sampling step, $n = (s-a)/\Delta t$, and

$$\alpha_i \approx 1 - e^{-\Delta t\ \rho_\alpha(f(i\,\Delta t))} \tag{3}$$

Discretizing the integral on $[a,b]$ in Equation 1 as a summation, we have the following discrete approximation for I,

$$I \approx \sum_{i=0}^{n} \rho_E(i) \prod_{j=0}^{i-1}(1-\alpha_j) \tag{4}$$

where $\rho_E(i) = \rho_E(f(i\,\Delta t))$ is given by the transfer function. Evaluating the transfer function after reconstruction is known as postclassification. Typical sampling behavior of postclassification with uniform sampling along the ray is illustrated in Figure 1(a). When high-frequency features are present in $\rho_\alpha(f(t))$, many samples are required to accurately integrate along the ray.

To eliminate artifacts and achieve high-quality volume rendering, we must adequately sample with respect to the Nyquist limits of all component functions contributing to the signal. The principal signal sources consist of the scalar field function $f(\vec{R}(t))$ and the transfer function $\rho_\alpha(f(\vec{R}(t)))$. Engel et al. [5] note that this frequency can be either the maximum Nyquist frequency of all separate sources, or the product of the Nyquist frequencies of these sources. By discretizing the transfer function and scalar field integrals separately, preintegration can achieve greater fidelity for high-frequency transfer functions with fewer samples, as illustrated in Figure 1(b).

Separately integrating the transfer and field functions via preintegration presents separate issues, however. Problems occur when the scalar field or transfer function are undersampled by their respective discrete integrations. Like postclassification, preintegration is susceptible to undersampling, though artifacts are manifested differently. Preintegration assumes the scalar field function varies piecewise-smoothly between entry and exit samples, $f_i = f(\underline{t})$ and $f_o = f(\bar{t})$. Depending on the frequency of the field function, this is often not the case. Specifically, computing the opacity integral on a segment uses the trapezoid rule (or similar numerical integration), which scales the opacity summation by $\Delta f = |f_i - f_o|$ to approximate ρ_α. When $\rho_\alpha(f)$ is smooth (specifically, Lipschitz) this approximation behaves nicely. However, sharp features in the transfer function break this assumption, leading to bias and improperly scaled opacity. Though blending the integrals of front and back samples smoothens results [21], it does not accurately capture sharp peaks. In addition, preintegration relies on a fixed quantization of entry and exit opacities into a table. Permitting dynamic changes in the transfer function limits the size of this table, hence the minimum width Δf between two field values used to query the transfer function integral. Nonetheless, visualizing features with higher precision can be desirable for more accurate classification.

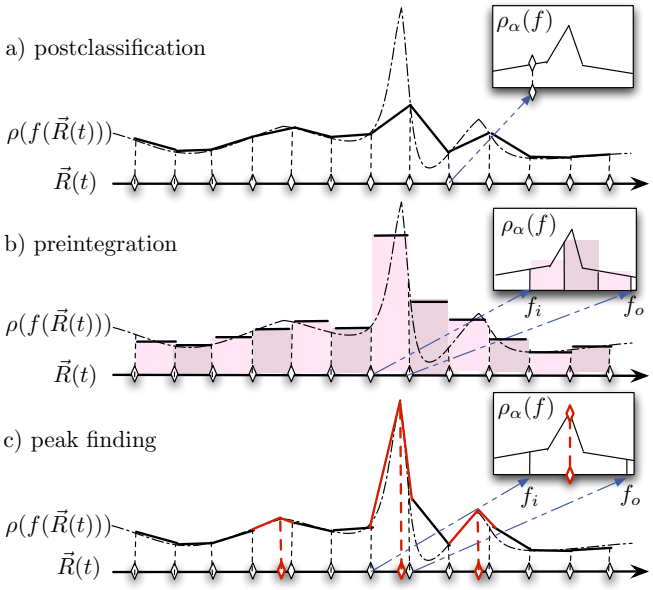

Fig. 1. Integration methods for direct volume rendering.

This paper describes two techniques that overcome deficiencies of existing methods. The main contribution is peak finding, which overcomes many limitations of postclassification and preintegration by sampling directly at sharp features of a transfer function. This consists of analyzing a transfer function for local maxima, and explicitly solving for roots of the filter function to render isosurfaces at these peaks. As with preintegration, peak finding employs a 2D lookup table; however rather than querying an approximation of the integral itself, we query which peaks possibly lie within that range of field values. The general concept is illustrated in Figure 1(c), and its implementation is described in detail in Section 4.

In section 5 we present differential sampling. We note that transformations on the ray from world-space to image-space convolve the volume rendering integral, and provide a new sampling method respecting the Nyquist frequency of the image plane. Our method borrows from the ray differentials formulations of Igehy [8] in developing its sampling strategy. This is discussed in Section 5.

Our system consists of a straightforward volumetric ray caster, employing a grid acceleration structure traversed per-ray in a GLSL fragment shader, and classifying via a 1D transfer function specified as a piecewise-linear set of points. Section 6 discusses how to integrate differential sampling and peak finding into this framework. The end goal of this work is to enable interactive high-fidelity volume rendering with sharp transfer function features using fewer samples than conventional methods. We show how these algorithms help to accomplish that in Section 7.

4 PEAK FINDING

Peak finding is motivated by the shortcomings of both standard postclassified (Figure 1(a)) and preintegrated (Figure 1(b)) volume rendering with transfer functions containing sharp features approaching Dirac impulses. The general approach is similar to isosurface ray casting in that we solve directly for roots. Ray-isosurface intersection consists of solving the continuous reconstruction filter function as a 1D implicit function of t at an isovalue: $f(\vec{R}(t)) - \upsilon = 0$.

Numerous numerical methods exist for solving roots of this equation; interactive ray tracing algorithms commonly employ a combination of Descartes's rule of signs and an iterative solver [7, 22, 30], or more robust recursive methods such as interval arithmetic [12]. The substantial difference is that in these systems, the isovalue is given explicitly by the user; whereas in ours the isovalue must be inferred from the transfer function. By employing these root-finding methods in searching for peaks of the transfer function, we have far lesser chance of missing them, allowing for smoother reconstruction and better shading of isosurface features within our volume rendering framework. The general concept is illustrated in Figure 1(c).

4.1 Determining peaks and building the lookup table

Peak finding is similar to preintegration in that we query a 2D lookup table for each segment along the ray. However, rather than storing a preintegrated radiance approximation, our table stores an isovalue υ or set of isovalues υ_i that possibly exist within this segment, sorted from the first to last peak value encountered on a given segment defined by the entry and exit values of the scalar field function, $[f_i, f_o]$.

Before building the lookup table, we analyze our transfer function ρ_α and search for peaks. Specifically, we consider whether a given point is a local maximum (i.e. greater than both its immediate neighbors) with respect to the opacity component. The set of peaks consists of at most half the number of actual data points in our piecewise-linear transfer function, but typically it is far less. Smooth 1D functions such as splines would have relatively fewer peaks, existing at the critical points of these functions. As we are interested in sharp features, we consider piecewise-linear functions. It is equally possible to use this technique to search for local minima; however due to their low radiance contribution the impact of doing so is not generally noticeable.

Having computed the array of peaks, we construct the lookup table. For a range of values $[i, j]$ corresponding to lookup entries from our volume $f(\underline{t}), f(\bar{t})$. If $i < j$, we search our transfer function for the next peak point (or in the case of multiple peaks, next 4 points) such that the opacity $\rho_\alpha(\upsilon) > i$ and $\rho_\alpha(\upsilon) \leq j$. If $i > j$, we search in descending order for peaks with $\rho_\alpha(\upsilon) \leq i$ and $\rho_\alpha(\upsilon) > j$. When necessary, a segment spanning multiple peaks will reverse the sorting order to register all possible peaks within that segment. This process is again similar to preintegration, except that separate discrete peak values are stored instead of a single integral approximation. In each table entry, we store the domain isovalue(s) υ corresponding to each peak. When no peak exists, we use a flag outside of the range of scalar values in the volume. Building the lookup table is relatively undemanding, and proceeds in $O(N^2)$ time, similarly to the algorithm of [21] for preintegration. In practice, building a peak-finding table is roughly twice as fast as building a preintegrated table at the same resolution. Moreover, in many cases a coarser discretization (128 bins) is sufficient for peak finding, whereas preintegration would require a larger table for comparable quality when rendering near-discrete isosurfaces.

4.2 Root solving and classification

Peak finding occurs between samples in the main ray casting loop. Before sampling at the next step $\bar{t}$, we fetch the nearest peak value from a 2D texture using the same $f(\underline{t}), f(\bar{t})$. If the peak exists, we subtract that isovalue from the entry and exit values, and employ Descartes' rule of signs. If this test succeeds, we assume the segment contains a root. Bracketed by $\underline{t}$,$\bar{t}$, we use three iterations of a secant method (also employed by [7, 22]) to solve the root:

$$t_1 = t_0 - f(t_0)\frac{t_1 - t_0}{f(t_1) - f(t_0)} \tag{5}$$

When the secant method completes, we have an estimate for the root t along the ray segment. We now sample at this position and perform postclassification. However, sampling at the peak requires two subtle choices. First, we do not evaluate our field $f(\vec{R}(t))$, but rather assume that the value at this point is our desired isovalue. This works because we are solving for the root position, not its value; moreover for sharp transfer functions it is crucial in avoiding Moire patterns. Second, we do not not scale ρ_α by the segment distance Δt (in Equation 3) but instead use a constant $\Delta t = 1$. Although this may seem counterintuitive, the scaled extinction coefficient is itself a correction mechanism for the inherently discrete approximation of the volume rendering integral. Moreover, an unscaled opacity assumes that we always sample at this isovalue regardless of the sampling rate or local behavior of $\rho_\alpha(f)$ along the ray segment. This is precisely our goal with peak finding. While the resulting approach arguably biases the volume rendering integration towards these peaks, it is critical in detecting them without excessively increasing the sampling rate. In practice this strategy does not greatly bias our integral, as the relative contribution of values outside the peak is small.

Finding multiple peaks can be useful when the step size Δt is large, or when peaks are spaced closely together. Our implementation handles up to four multiple peaks within a single segment with a straightforward extension, which can be enabled at runtime as necessary. As described in Section 4.1 we construct the peak finding table with four sequential peaks contained within the given segment $[f_i, f_o]$. Since isosurfaces are encountered in precomputed order between the minimum and maximum field values, we can simply perform peak finding sequentially on all four values in that order.

4.3 Algorithm integration and usage

Peak finding is equivalent to volume rendering with a discrete isosurface-finding step in between. One can trivially modify the algorithm to support different rendering modalities. We allow for:

- Sampling from *both* uniformly/differentially sampled DVR and peak finding (default).
- Sampling from *either* uniformly/differentially sampled DVR or peak finding (`PEAK_XOR_DVR`).
- Transparent isosurfacing of peaks only (`PEAK_ONLY`).
- DVR only, disabling peak finding (`DVR_ONLY`).

These options can be invoked with small switches to the shader code and incur no performance penalty or code overhead. The uppercase flags above correspond to macros in the GLSL pseudocode provided in the Appendix.

Peak finding is attractive in that its algorithm is not significantly different from either volume rendering or isosurface ray casting. Both algorithms employ regular sampling, in the case of DVR to compute the volume rendering integral and in the case of isosurfacing to isolate roots. Peak finding takes advantage of this and does both. As a result, this technique can be implemented quickly by extending existing renderers. Although we propose peak finding in conjunction with differential sampling, the two techniques are orthogonal. It is equally possible to employ peak finding in a uniform sampling ray caster, a slice-based volume renderer, or a shear-warp system.

Overall, peak finding and preintegration are similar, but make different assumptions about the integral over a given segment. Preintegration assumes this integral can be accurately approximated by piecewise summation. This works well when the transfer function and convolved field are smooth, but encounters difficulties when they are not. Peak finding assumes this integral can be approximated by one or several discrete impulses. This introduces bias, but is better suited for noisy data and sharp C_0 transfer functions for which standard techniques fail.

5 Differential Sampling for Volume Rendering

Uniform sampling ignores an important component of the convolved volume rendering integral and its resulting Nyquist limit. With a pinhole camera, the projective transformation on the image plane is itself a signal convolution. Thus, regular sampling in world-space undersamples features close to the viewpoint relative to those further away. To remedy this, we can employ a sampling strategy that uses the ray distance itself as a sampling metric. This can be accomplished with a new function T whose derivative varies linearly with distance, i.e.

$$\Delta T = \frac{\partial T}{\partial t} = at + b \qquad T(t) = \frac{a}{2}t^2 + bt + c \tag{6}$$

Then we sample along the ray at $\vec{R}(T(t))$. The question remains how to choose a, b and c so that the sampling step is proportional to pixel width. We turn to the concept of ray differentials [8], which quantifies world-space transformations in image-space derivatives. Specifically, we use the ray differential transfer equation to formulate T as a function of image-space.

5.1 Ray differentials

With ray differentials [8], the general goal is to compute the image-space derivatives of a series of functions convolving the image plane, beginning with generation of rays in a pinhole camera,

$$\vec{d}(x,y) = \vec{w} + x\vec{u} + y\vec{v} \tag{7}$$

where $\vec{w}$ is the central view direction, $\vec{u}, \vec{v}$ are the right and up vectors. Unitizing a ray $\vec{r}(t) = \vec{o} + t\vec{d}$ comprises another transformation:

$$\vec{O}(x,y) = \vec{o} \qquad \vec{D}(x,y) = (\vec{d})(\vec{d}\cdot\vec{d})^{-1/2} \tag{8}$$

Then the unit-parameterized ray $\vec{R}(t) = \vec{O} + \vec{D}t$ has the image-space partial with respect to x (and similarly for y):

$$\frac{\partial \vec{R}}{\partial x}(t) = \frac{\partial \vec{O}}{\partial x} + t\frac{\partial \vec{D}}{\partial x} + \frac{\partial t}{\partial x}\vec{D} \tag{9}$$

As $\frac{\partial \vec{O}}{\partial x} = 0$, this holds for any discrete difference Δt as well. For our purposes of choosing a constant image-space measure, it suffices to consider only x differentials. Lastly, the differential of the unitized $\vec{D}$ with respect to the x image-space coordinate is:

$$\frac{\partial \vec{D}}{\partial x} = \frac{(\vec{d}\cdot\vec{d})\vec{u} - (\vec{d}\cdot\vec{u})\vec{d}}{(\vec{d}\cdot\vec{d})^{3/2}} \tag{10}$$

Derivations are given in more detail in the original paper [8].

5.2 Differential sampling construction

Our general strategy is to define a base sampling rate proportional to an image-space quantity, and use the ray differential transfer equation (Equation 9) to derive our sampling function T. To accomplish this, we use the image-space x as our discretization, and construct a sampling scheme where $\frac{\partial T}{\partial x}$ is proportional to the differential quantity $\frac{\partial \vec{R}}{\partial x}(\Delta t)$. As world-space Δt is proportional to x, for some scalar k,

$$\Delta t = kx \qquad \frac{\partial \Delta t}{\partial x} = k \tag{11}$$

Since $\vec{D}$ is normalized and our discrete step Δt is arbitrary, the user can choose any k and preserve a correlation between the distance-based sampling step Δt and x. Similarly, to use x as unit of measure along the ray, we project $\frac{\partial \vec{D}}{\partial x}$ so that it is collinear with $\vec{D}$, i.e. $\frac{\partial \vec{D}}{\partial x} = |\frac{\partial \vec{D}}{\partial x}|\vec{D}$. Then from Equation 9, we have:

$$\begin{aligned}\frac{\partial \vec{R}}{\partial x}(\Delta t) &= \Delta t\frac{\partial \vec{D}}{\partial x} + \frac{\partial \Delta t}{\partial x}\vec{D} = kx|\frac{\partial \vec{D}}{\partial x}|\vec{D} + k\vec{D} \\ &= \vec{D}(k|\frac{\partial \vec{D}}{\partial x}|x + k)\end{aligned} \tag{12}$$

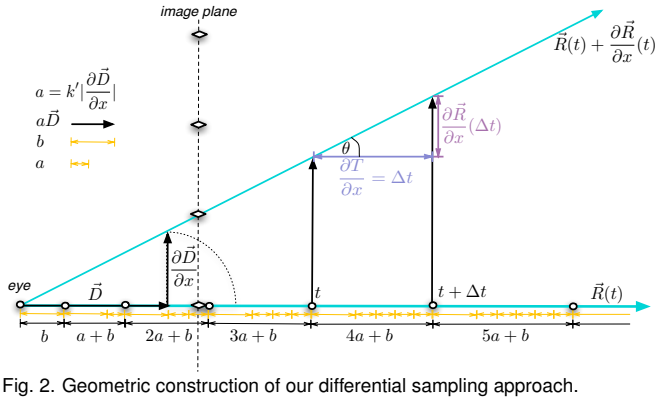

Fig. 2. Geometric construction of our differential sampling approach.

Since $|\vec{D}| = 1$, this gives us

$$|\frac{\partial \vec{R}}{\partial x}(\Delta t)| = k|\frac{\partial \vec{D}}{\partial x}|x + k \tag{13}$$

From Figure 2, notice that $|\frac{\partial \vec{R}}{\partial x}(\Delta t)| = \tan(\theta)\Delta t$. Since θ between any two rays is constant, $\tan(\theta)$ is also constant (its computation is left as an exercise). This can be incorporated into a new constant $k' = k\tan(\theta)$; or if k is arbitrarily chosen we can omit this step and use $k' = k$. We then employ the differential construction of T in Equation 6 but in terms of image-space x,

$$\frac{\partial T}{\partial x} = \Delta t = k'|\frac{\partial \vec{D}}{\partial x}|x + k' \tag{14}$$

For convenience let $a = k'|\frac{\partial \vec{D}}{\partial x}|$ and $b = k'$. The antiderivative yields our differential sampling function T:

$$\frac{\partial^2 T}{\partial x^2} = a \qquad \frac{\partial T}{\partial x} = ax + b \qquad T(x) = \frac{a}{2}x^2 + bx + c \tag{15}$$

When we begin sampling at $t = T(x) = 0$, we can assume $c = 0$.

5.3 Computing and incrementing samples

Differential sampling is simple to implement in a volume ray casting framework. We first compute $|\frac{\partial \vec{D}}{\partial x}|$ from Equation 10. While the user can choose any k, we ensure it is some multiple of world-space pixel footprint at the image plane, e.g. $k = s_k|\vec{u}\Delta x + \vec{v}\Delta y|$. From this we compute k' (if necessary), a and b. Theoretically, $s_k < 1/2$ is required to satisfy the Nyquist limit of the image plane. In practice this rate is excessive, and $s_k = 4$ is a good conservative default.

From the ray origin, the sampling process begins at $x = 0$, where

$$\frac{\partial T}{\partial x} = b \qquad T(x) = 0 \tag{16}$$

Then at each ray casting iteration, we sample at $\vec{P} = \vec{P}(T(x))$, and perform the following increments, where Δt is our discretization of $\frac{\partial T}{\partial x}$,

$$\vec{P}_1 = \vec{P}_0 + \Delta t_0\,\vec{D} \qquad \Delta t_1 = \Delta t_0 + a \tag{17}$$

Thus, incrementing the position from one sample to the next consists only of an extra vector multiplication and addition, on top of the vector addition for uniform sampling. This is also outlined in the pseudocode in the Appendix.

6 Implementation

We implemented our ray casting framework in OpenGL and GLSL. The pinhole camera vectors $\vec{w}$, $\vec{u}$ and $\vec{v}$ are computed on the CPU and then sent to the fragment shader, where a ray is generated from the pixel x and y values according to Equation 7. The 1D transfer function is given as a set of points $\{v, \{r,g,b,a\}\}$, then processed into a fairly wide (8K elements) 1D texture, allowing for rapid access on the GPU and generally sufficient transfer function precision $\Delta f > 1e-4$. We implemented a tricubic B-spline filter using the method of [29], with the BC smoothing ($B = 2, C = 1$) kernel of [23]. We optionally employ this for both DVR sampling and root solving.

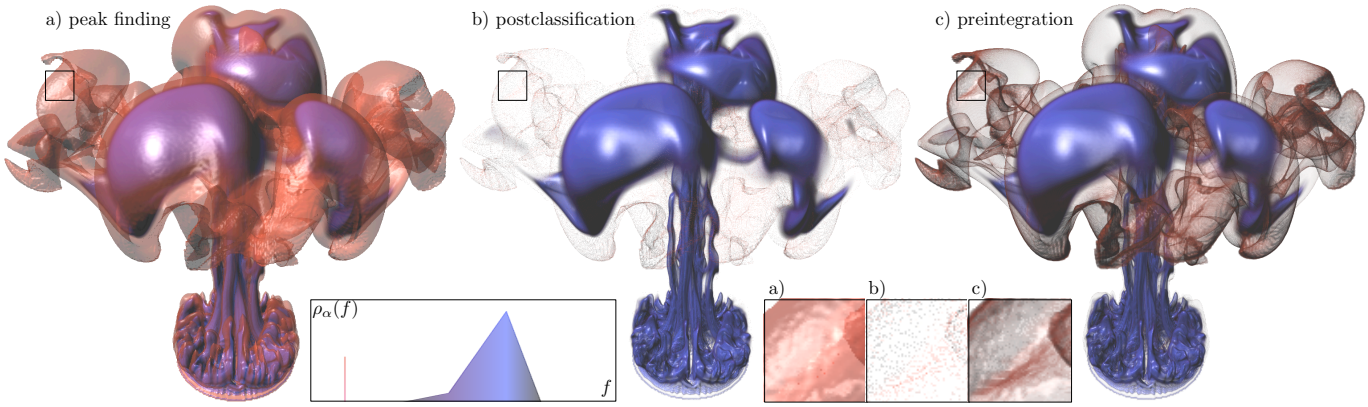

Fig. 3. Simulated temperature of a heptane fire, with a transfer function consisting of a near-Dirac peak (width $\Delta f < 1e-4$) in red, and a smoother feature for contrast in blue. Peak finding, postclassification and preintegration render at 7.1, 11.6 and 8.8 fps, respectively, at 1024x900.

6.1 Space skipping

Even with fairly dense transfer functions, most data sets are sparse enough to warrant an empty space skipping mechanism. We choose a simple uniform grid with a 3DDDA algorithm [1] where each grid cell stores min-max values of enclosed voxels. Fairly coarse grids (64^3 cells) work best on the GPU, and this structure can be updated interactively when the transfer function changes. The fragment shader then traverses the macrocell grid using the 3DDDA algorithm in an outer loop. When a macrocell is nonempty, we enter the volume rendering loop, with peak finding tests taken between samples. To begin sampling, we find the first t at which to sample when entering a macrocell. With differential sampling, we solve for the maximum x after T_{enter},

$$ax^2/2+bx = T_{enter} \qquad x_{tenter} = (-b+\sqrt{b^2+2aT_{enter}})/a \qquad (18)$$

We then compute the floor values $\lfloor x_{tenter} \rfloor$, $T(\lfloor x_{tenter} \rfloor)$ and $\frac{\partial T}{\partial x}(\lfloor x_{tenter} \rfloor)$, which can be simplified significantly from Equation 15; and subsequently sample and increment as in Equation 17. To avoid duplicate samples, we store the greatest t at which we already sampled, and use the maximum of that and T_{enter}.

6.2 Adaptive sampling

As discussed in [7], purely adaptive methods (for example based on local gradient) perform poorly on GPUs due to poor thread coherence. However, we do achieve better performance by varying the sampling rate on a per-macrocell basis. In this scheme, each macrocell computes a metric based on the ratio of the maximum standard deviation of its voxels to that of the entire volume, $m = \sqrt{\lceil \mathrm{Var}(f_{cell}) \rceil / \lceil \mathrm{Var}(f_{vol}) \rceil}$. As this represents a multiplier for the frequency, its inverse can be used to vary the sampling step size Δt. In practice we wish this to be a positive integer, and a multiplier $M = 2m^{-1}+1$ delivers good results. With uniform sampling one simply employs $M\Delta t$ as the new sampling rate. With differential sampling, M modifies our increments as follows:

$$\frac{\partial^2 T_M}{\partial x^2} = \sum_{i=1}^{M} a = \frac{M\,(M+1)}{2} a \qquad \frac{\partial T_M}{\partial x} = Max + \frac{\partial^2 T_M}{\partial x^2} + b \qquad (19)$$

No modifications to $T(x)$ are required, since the initial x for that macrocell can be any integer.

7 Results

Unless otherwise stated, all results were collected on a 2.5 GHz Intel Xeon and an NVIDIA 285 GTX GPU, with trilinear filtering, differential sampling ($s_k = 4$) and the exclusive-or peak finding modality. For each scene we plot the $(f, \rho_\alpha(f))$, scaled to the maximum opacity of the transfer function. To evaluate complexity, we count the total number of filter evaluations (including peak finding) or DVR-only samples (without peak finding), and divide these by the number of pixels. As with any DVR system, performance varies widely with the number of samples taken. Opaque isosurfaces and low-frequency scenes are simplest and render at real-time rates. The focus of our work is in handling sharp features, which requires higher sampling rates. Overall, image quality is excellent and our system is generally interactive (Table 1). While analysis of macrocells falls outside the scope of this paper, they usually deliver 1.2x to 5x performance improvement depending on the scene. Although other approaches have greater total sample throughput, our system is competitive in how it spends samples and resulting quality.

dataset	dimensions	samp/r	dvrs/r	fps – pf / pc / pi
heptane (f. 3)	302x302x302	190	120	7.1 / 11.6 / 8.8
(f. 5b)	302x302x302	117	58	8.2 / 17.0 / 12.3
(f. 5c)	302x302x302	230	54	10.6 / 18.0 / 12.7
zebrafish (f. 4)	900x500x930	1030	165	2.1 / 2.0 / 1.7
aneurism (f. 6)	256x256x256	561	342	5.3 / 8.6 / 7.0
+BS filter (f. 7)	256x256x256	157	82	1.8 / 2.7 / 2.5
fireset (f. 6)	512x256x512	336	193	6.6 / 9.5 / 7.3
backpack (f. 7)	512x512x373	1078	633	2.1 / 2.9 / 2.1

Table 1. Overall performance in frames per second and average samples per ray for selected scenes with differential sampling $s_k = 4$. The right three columns show average samples (filter function evaluations) per ray, average DVR-only samples per ray, and fps with peak finding, postclassification, and preintegration.

7.1 Peak finding

Peak finding is useful when the combined frequency of the volume and transfer function is too high for effective regular sampling. In such cases, postclassification would require near-infinite sampling to accurately reproduce features. Preintegration succeeds in detecting high-frequencies of the transfer function, but integrates and shades them incorrectly when undersampling the scalar field.

An obvious scenario in which conventional sampling methods fail is a transfer function containing one or more Dirac-like features, as shown in Figure 3. Peak finding succeeds in reproducing these features as semi-transparent isosurfaces, and rendering smoother volumetric features in the correct order. While postclassification misses peak features outright, preintegration detects and reproduces a surface. However, with preintegration the range Δf along a given segment can significantly skew the opacity integral; two segments with different Δf may sample the same impulse but have different irradiances. With peak finding, this is not the case. In addition, preintegration shades at the segment endpoint, as opposed to locally at the hit position of the isosurface, resulting in Moire patterns. Finally, when an impulse is defined with a discretization smaller than that of the preintegrated table, peak finding with a smaller table can reproduce features that preintegration misses. In practice, this is less a concern than the aforementioned integration and shading issues with preintegration.

Peak finding is an intriguing method for rendering noisy or entropic data, for example from scanned sources in medicine or biology. Here, even when the transfer function is sampled adequately, the filtered field function of the volume (hence the convolved signal) is not. While artifacts are not as noticeable due to the noisy nature of renderings, high-frequency features are again omitted. Due to convolution of the high

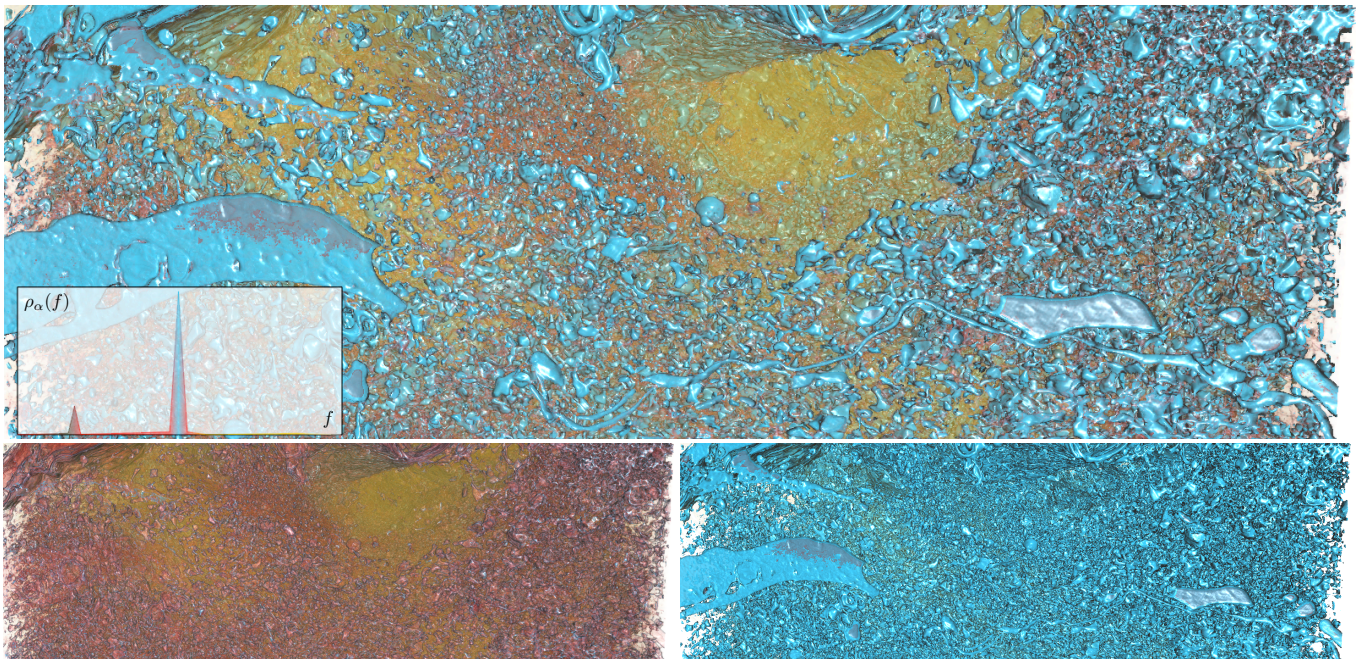

Fig. 4. Zebrafish optic tract acquired through electron microscopy [9] rendered with differential sampling and peak finding at 1600x512 resolution. Peak finding (top, 2.1 fps) enables better classification of narrow-band segments in entropic data. Preintegration (bottom left, 2.0 fps) has difficulty accurately reproducing such features, and semi-transparent isosurfacing (bottom right, 4.3 fps) lacks the depth cues provided by volume rendering.

data frequency, features can be lost even with moderate-frequency transfer functions. Simply increasing opacity at peaks does not correct the problem, and widening the transfer function broadens the classification. Choosing a higher sampling rate can remedy this, but at high performance cost. Meanwhile, at sampling rates well below the Nyquist limit, peak finding successfully reproduces sharp features with the desired opacity and color, as shown in Figure 4. The fireset in Figure 6 also illustrates this phenomenon.

Finding multiple peaks is typically not necessary unless several sharp features are close together in the transfer function. This option better ensures peaks are rendered in the correct order, and costs roughly 20% performance (Figure 5 (left)). More significantly, we find that bias from always sampling at peaks is manageable. Figure 5 (right) considers a smooth transfer function that looks nearly identical with peak finding and postclassification (Figure 5c,e). Peaks with opacity magnified by 16 (Figure 5d) and peak isosurfaces only (Figure 5f) are shown for contrast. The only disadvantage of peak finding in such cases is that it is not necessary and more costly. While it is possible to construct transfer functions for which peak regions have relatively higher contribution to the radiance and show greater bias, for the most part peak finding accentuates isosurface-like features as desired.

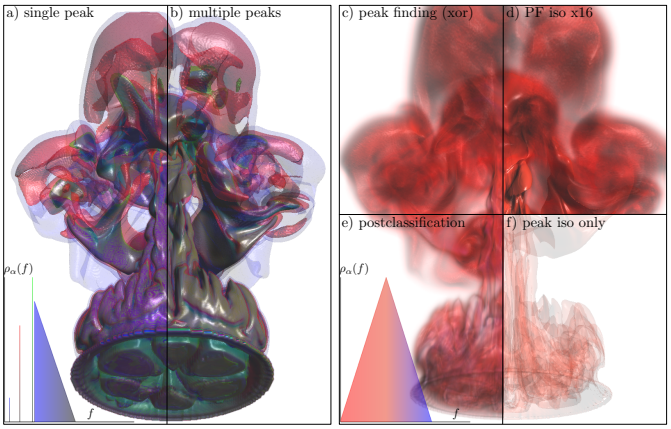

Fig. 5. Peak finding behavior (800x1024 resolution). Left: finding single and multiple peaks, at 10.7 and 8.2 fps, respectively. Right: bias from always sampling at peaks is generally subtle. At full frame resolution, (c-f) render at 11.7, 13.6, 20.5 and 15.6 fps, respectively.

7.2 Differential sampling

Differential sampling delivers better results close to the viewpoint, and not noticeably worse quality in the distance. A major appeal of this method is that the sampling rate is view-dependent; it automatically and locally matches sampling to the frequency of the image plane, thus requiring less work on the part of the user. In evaluating differential sampling, it is difficult to enforce a constant average sampling rate, so we use frame rate as the control variable and compare the results in Figure 6. Exact performance figures are given in Table 2. At similar frame rate, uniform sampling undersamples nearby features, and differential sampling remedies this, yielding consistently better quality and surpsingly little quality loss further away. Peak finding amplifies undersampling artifacts at silhouettes; as a result differential sampling in conjunction with peak finding is particularly desirable up close.

More subjectively, we can choose a single converged image as the control, and compare frame rates required for each scheme to achieve comparable quality. We use Figure 7 and the differential sampling halves of Figure 6 as reference; results are given in Table 2 (bottom). Adequately sampled, these scenes look generally similar with uniform and differential schemes. However, differential sampling can deliver up to 3x better frame rate, particularly when overall frequency is low. In Figure 7(a,b), converged images of the aneurism with postclassification and B-spline filtering look nearly identical, but run at 1.0 and 2.7 fps with uniform and differential sampling, respectively (0.86 and 1.8 fps with peak finding). Conversely, in cases where data is entropic and classified with multiple peaks, differential sampling is less effective, requiring a smaller s_k to adequately sample faraway regions, while oversampling nearby features. This is more noticeable with peak finding, where adequate sampling is necessary for robust root isolation of isosurfaces. Overall, differential sampling seldom delivers worse quality than uniform at the same frame rate. The backpack in Figure 7(c,d), a noisy scanned volume classified with peak finding and multiple peaks, still renders at 1.6 fps with both sampling methods and similar quality.

As evident in Table 2, differential sampling often requires half or less as many uniform samples for equivalent visual quality. Ideally, half as many samples would correspond to exactly double the frame rate. In practice this is not the case, due to the parallel nature of GPUs and worse memory coherence at far-away samples when using differ-

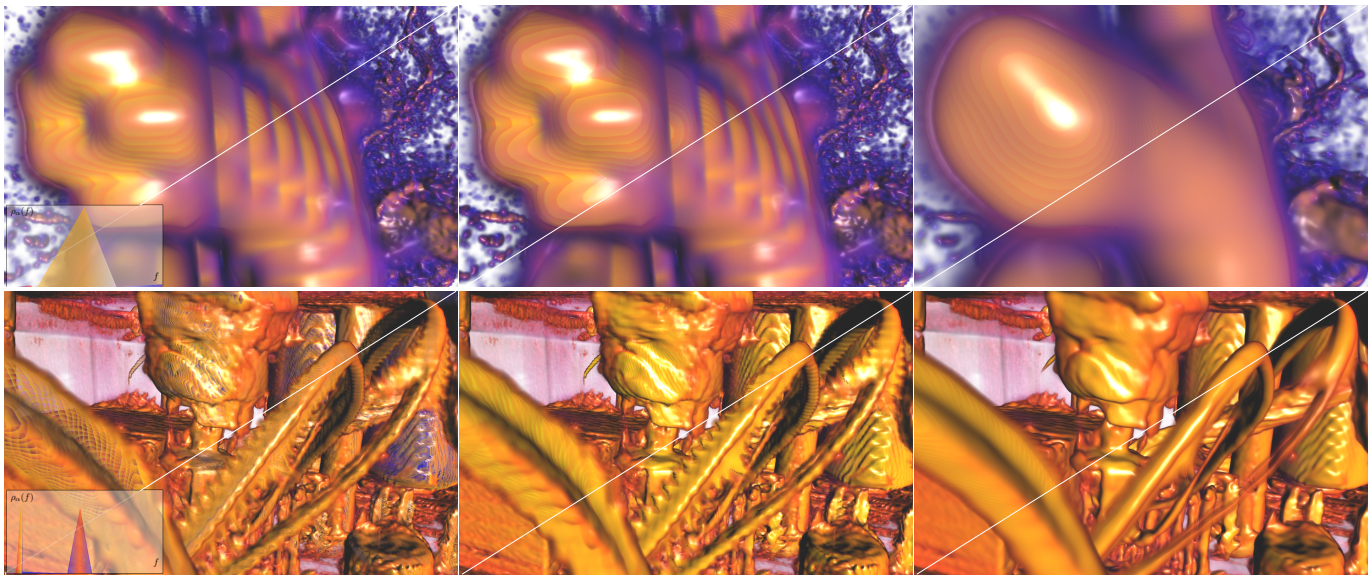

Fig. 6. Close-up scenes with uniform (left) and differential sampling (right) at similar frame rates, rendered at 1280x800. Columns show postclassification, peak finding, and peak finding with higher-order B-spline filtering. Aneurism and fireset scenes are shown in the top and bottom rows, respectively.

ential sampling. With tricubic B-spline filtering, the higher cost of computing samples outweighs this penalty, yielding relatively better performance with differential sampling than with uniform (1.5-3x as opposed to 1-2x). Nonetheless, differential sampling remains clearly worthwhile with trilinear filtering.

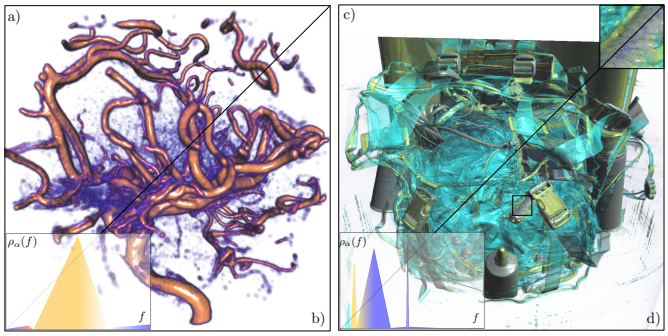

Fig. 7. Far views with differential sampling (b,d) render 1-3x faster than uniform sampling (a,c) at similar quality. Left: aneurism (postclassified with B-spline filtering). Right: noisy backpack data. (a,b,c,d) render at 1.0, 2.7, 1.6 and 1.6 fps, respectively at 1024^2.

dataset	postclassification	peak finding	pf+B-spline
Frame rate held constant – Fig. 6			
aneurism–close (Fig. 6)	uni./diff.	uni./diff.	uni./diff.
fps	7.4 / 8.7	5.6 / 5.3	1.0 / 1.1
samples/ray	361 / 342	580 / 561	770 / 827
fireset–close (Fig. 6)	uni./diff.	uni./diff.	uni./diff.
fps	4.8 / 5.1	3.6 / 4.1	0.96 / 0.96
samples/ray	396 / 380	509 / 528	408 / 494
Converged sampling quality – Figs. 6(right halves) and 7			
aneurism–close (Fig. 6)	uni./diff.	uni./diff.	uni./diff.
fps	2.0 / 6.4	2.4 / 4.2	0.42 / 0.80
samples/ray	1433 / 483	1419 / 700	1854 / 964
fireset–close (Fig. 6)	uni./diff.	uni./diff.	uni./diff.
fps	1.1 / 3.1	2.2 / 4.6	0.42 / 0.77
samples/ray	2123 / 772	1441 / 527	1568/ 576
aneurism–far (Fig. 7)	uni./diff.	uni./diff.	uni./diff.
fps	5.8 / 13	4.3 / 7.1	0.86 / 1.8
samples/ray	343 / 113	471 / 174	372 / 157
backpack–far (Fig. 7)	uni./diff.	uni./diff.	uni./diff.
fps	2.7 / 2.4	1.6 / 1.6	0.36 / 0.43
samples/ray	640 / 748	1794 / 1865	822 / 720

Table 2. Differential sampling performance for images in Figs. 6 and 7.

8 Discussion

Our proposed techniques advance the state-of-the-art in high-quality volume ray casting. Peak finding allows for near-discrete isosurfaces to be specified within a volume rendering transfer function, and provides a new tool in the classification arsenal. It yields viable classification of entropic and noisy data, handles pathological cases that are unadressed by postclassification and preintegration, and is not significantly slower than those techniques. Differential sampling allows for better quality rendering of features closer to the camera, with less overall sampling and correspondingly higher frame rate.

The main drawback of peak finding is that it is more costly than preintegration, and unnecessary when the transfer function and data are smooth. Again, an argument can be made that introducing discrete isosurfaces into the volume rendering integral is inherently biased. In addition, the rule of signs is not a robust root isolation method, and surfaces can be missed near sharp silhouettes. The main limitation of differential sampling is that it would be difficult to implement outside of a ray casting framework. When s_k is very small, differential sampling encounters numerical problems resulting in worse artifacts at greater sampling rates, shown in the close-up in Figure 7(c,d). This is rarely an issue in practice, and could be remedied with double-precision GPU arithmetic. The chief drawback of our implementation is that it traverses an acceleration structure in the fragment shader, which is likely slower than rasterized bricking or slicing. Most of our chosen scenes are costly to sample regardless of space skipping, but we could employ a proxy rasterization technique such as [17] for better performance.

Several extensions to this work are worth pursuing. Differential sampling could be used in more traditional applications of ray differentials such as multiscale filtering and level of detail, which could improve quality and allow efficient rendering of large data. Peak finding could be extended to handle multidimensional and multifield transfer functions, which could use topological methods to find peaks in higher dimensions. We are also interested in combining preintegration and peak finding for better classification.

Acknowledgements

This work was supported by the German Research Foundation (DFG) through the University of Kaiserslautern International Research Training Group (IRTG 1131); as well as the National Science Foundation under grants CNS-0615194, CNS-0551724, CCF-0541113, IIS-0513212, and DOE VACET SciDAC, KAUST GRP KUS-C1-016-04. Additional thanks to Liz Jurrus and Tolga Tasdizen for the zebrafish data, and to the anonymous reviewers for their comments.

REFERENCES

[1] J. Amanatides and A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Proc. EG 87*, pages 3–10. Eurographics Association, 1987.

[2] B. Cabral, N. Cam, and J. Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization*, pages 91–98, New York, NY, USA, 1994. ACM Press.

[3] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In J. Rokne, W. Wang, and R. Klein, editors, *Proceedings of Pacific Graphics 2003*, pages 325–334, Oct. 8–10 2003.

[4] T. J. Cullip and U. Neumann. Accelerating Volume Reconstruction With 3D Texture Hardware. Technical report, 1994.

[5] K. Engel, M. Kraus, and T. Ertl. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 9–16. ACM New York, NY, USA, 2001.

[6] M. Hadwiger, A. Kratz, C. Sigg, and K. Bühler. GPU-accelerated Deep Shadow Maps for Direct Volume Rendering. *Graphics Hardware*, 6:49–52, 2006.

[7] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[8] H. Igehy. Tracing Ray Differentials. In *Computer Graphics (Proceedings of ACM SIGGRAPH)*, pages 179–186, 1999.

[9] E. Jurrus, M. Hardy, T. Tasdizen, P. Fletcher, P. Koshevoy, C. Chien, W. Denk, and R. Whitaker. Axon tracking in serial block-face scanning electron microscopy. *Medical Image Analysis*, 13(1):180–188, 2009.

[10] J. Kniss, S. Premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. Gaussian transfer functions for multi-field volume visualization. In *Proceedings of IEEE Visualization*, pages 497–504, Oct. 2003.

[11] G. Knittel. The ULTRAVIS System. In *Proceedings of the 2000 IEEE symposium on Volume visualization*, pages 71–79. ACM Press, 2000.

[12] A. Knoll, Y. Hijazi, A. Kensler, M. Schott, C. Hansen, and H. Hagen. Fast Ray Tracing of Arbitrary Implicit Surfaces with Interval and Affine Arithmetic. *Computer Graphics Forum*, 28(1):26–40, 2009.

[13] A. Knoll, I. Wald, and C. Hansen. Coherent multiresolution isosurface ray tracing. *The Visual Computer*, 25(3):209–225, 2009.

[14] J. Krüger and R. Westermann. Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of IEEE Visualization 2003*, pages 287–292, 2003.

[15] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 451–458, New York, NY, USA, 1994. ACM Press.

[16] M. Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, 1988.

[17] B. Liu, G. Clapworthy, and F. Dong. Accelerating volume raycasting using proxy spheres. *Computer Graphics Forum (Proceedings of Eurovis 2009)*, 28(3):839–846, 2009.

[18] Y. Livnat and C. D. Hansen. View Dependent Isosurface Extraction. In *Proceedings of IEEE Visualization '98*, pages 175–180. IEEE Computer Society, Oct. 1998.

[19] Y. Livnat and X. Tricoche. Interactive point based isosurface extraction. In *Proceedings of IEEE Visualization 2004*, pages 457–464, 2004.

[20] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 21(4):163–169, 1987.

[21] E. Lum, B. Wilson, and K. Ma. High-quality lighting and efficient pre-integration for volume rendering. In *Proceedings Joint Eurographics-IEEE TVCG Symposium on Visualization 2004 (VisSym 04)*, pages 25–34. Citeseer, 2004.

[22] G. Marmitt, H. Friedrich, A. Kleer, I. Wald, and P. Slusallek. Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing. In *Proceedings of Vision, Modeling, and Visualization (VMV)*, pages 429–435, 2004.

[23] D. Mitchell and A. Netravali. Reconstruction filters in computer-graphics. *ACM Siggraph Computer Graphics*, 22(4):221–228, 1988.

[24] S. Parker, M. Parker, Y. Livnat, P.-P. Sloan, C. Hansen, and P. Shirley. Interactive ray tracing for volume visualization. *IEEE Computer Graphics and Applications*, 5(3):238–250, 1999.

[25] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive Ray Tracing for Isosurface Rendering. In *IEEE Visualization '98*, pages 233–238, October 1998.

[26] S. Röttger, S. Guthe, D. Weiskopf, T. Ertl, and W. Strasser. Smart hardware-accelerated volume rendering. In *VISSYM '03: Proceedings of the symposium on Data visualisation 2003*, pages 231–238, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[27] S. Röttger, M. Kraus, and T. Ertl. Hardware-accelerated volume and isosurface rendering based on cell-projection. In *Proceedings of IEEE Visualization*, pages 109–116. IEEE Computer Society Press Los Alamitos, CA, USA, 2000.

[28] J. Schreiner, C. Scheidegger, and C. Silva. High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1205–1212, 2006.

[29] C. Sigg and M. Hadwiger. Fast third-order texture filtering. *GPU Gems*, 2:313–329, 2005.

[30] J. M. Singh and P. Narayanan. Real-Time Ray Tracing of Implicit Surfaces on the GPU. Technical report, International Institute of Information Technology, Hyderabad, India, 2007.

[31] M. Sramek. Fast surface rendering from raster data by voxel traversal usingchessboard distance. *Proceedings of IEEE Visualization 1994*, pages 188–195, 1994.

[32] L. Westover. Footprint evaluation for volume rendering. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 367–376, New York, NY, USA, 1990. ACM Press.

[33] J. Wilhelms and A. Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[34] Y. Zhou and M. Garland. Interactive point-based rendering of higher-order tetrahedral data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1229–1236, 2006.

[35] M. Zwicker, H. Pfister, J. van Baar, and M. Gross. EWA volume splatting. In *Proceedings of IEEE Visualization*, pages 29–36, 2001.

Algorithm 1 Differential sampling and peak finding pseudocode.

```
vec4 ps0, ps1; float fs0, fs1, dt, iso; //globals

void peak_find(){
  float f0 = fs0 - iso;
  float f1 = fs1 - iso;
  if (f0 * f1 < 0){
    vec4 p0 = ps0; //(x,y,z,t) at segment entry
    vec4 p1 = ps1; //(x,y,z,t) at segment exit
    for(int k=0; k<2; k++){
      vec4 prt = p0 - (p1 - p0) * f0 / (f1 - f0);
      float frt = volume_filter(pnew.xyz) - iso;
      if (fnew * f0 < 0){
        p1 = prt; f1 = frt; }
      else{
        p0 = prt; f0 = frt; }
    }
    prt = p0 - (p1 - p0) * f0 / (f1 - f0);
    frt = iso;
    dt = 1.0;  //scale the opacity
    dvr_sample(prt, frt, dt);
  }
}
void raycast(){
  dt = b;
  ps1.w = 0; //t
  ps1.xyz = ray.origin;
  fs1 = volume_filter(ps1.xyz);
  for(;;){
    ps0 = ps1; fs0 = fs1;
    ps1 += dt * ray.direction;
#if DIFFERENTIAL_SAMPLNG
    dt += a;
#endif
    fs1 = volume_filter(ps1.xyz);
#if !DVR_ONLY
    iso = texture2D(peakTable, float2(fs0, fs1));
    if (iso != invalid_flag)
      peak_find();
#endif
#if !PEAK_ONLY
#if PEAK_XOR_DVR
     else
#endif
      dvr_sample(ps1, fs1, dt);
#endif
#endif
    if (color.a > termination_alpha) break;
  }
}
```