Accurate Volume Rendering based on Adaptive Numerical Integration

Leonardo Quatrin Campagnolo, Waldemar Celes
Tecgraf Institute and Department of Computer Science
PUC-Rio, Brazil
{lquatrin,celes}@tecgraf.puc-rio.br

Luiz Henrique de Figueiredo
IMPA – Instituto Nacional de Matemática Pura e Aplicada
Rio de Janeiro, Brazil
lhf@impa.br

***Abstract*—We present an adaptive integration strategy to evaluate the volume rendering integral for regular volumes. We discuss different strategies to control the step size for both the inner and the outer integrals in the volume rendering equation. We report a set of computational experiments that compare both accuracy and efficiency of our proposal against Riemann summation with uniform step size. The comparisons are made for both CPU and GPU implementations and show that our method delivers both accuracy control and competitive performance.**

***Keywords*-volume rendering; adaptive integration; error control; Simpson's rule;**

I. Introduction

Volume rendering is a widely-used technique to visualize scalar fields in three-dimensional datasets. The technique was first presented by Drebin [1] to render a set of medical images containing a mixture of materials.

The illumination model typically used for volume rendering is the well-known low albedo emission and absorption model presented by Max [2] and later described and simplified by Engel [3]. The ray casting algorithm for volume visualization renders the image by integrating the effects of light properties along each viewing ray [2], resulting in the well-known Volume Rendering Integral (VRI):

$$I = \int_0^D C(s(t))\,\tau(s(t))\,\exp\left(-\int_0^t \tau(s(t'))\,dt'\right)dt \qquad (1)$$

where D is the ray's length, C is the emitted light (given by the transfer function), τ is light extinction coefficient (also given by the transfer function), and s is the scalar field along the ray, which is parametrized by t. One of the main challenges in volume rendering is how to compute the VRI accurately, while maintaining good performance.

The VRI can be solved analytically only if severe constraints are imposed on the scalar field and on the transfer function. In practice, the integral can be solved only with numerical approximation methods. This raises the challenge of evaluating the error of the chosen integration procedure: this error should be kept within an acceptable limit to ensure accuracy.

It is common practice to simply use uniform steps to evaluate the integral as a Riemann sum. This makes implementations straightforward, specially on GPU architectures, but it lacks control: choosing a large step results in fast but not necessarily accurate solutions; choosing a small step to improve accuracy usually compromises performance.

Adaptive numerical integration methods appear as a solution to this dilemma. In these methods, the step size is adapted according to the estimated numerical error. This allows better control on accuracy and makes it possible to use strategies that automatically balance accuracy and efficiency. However, using adaptive methods to compute the VRI is not straightforward. We see two main challenges:

- The volume rendering equation involves two integrals: the inner one to compute transparency and the outer one to capture the final contribution. How can we employ adaptive step sizes for both integrals while preserving efficiency?
- Adaptive integration methods are typically implemented recursively. How can we map such recursive solutions to GPU architectures?

In this paper, we propose an adaptive integration algorithm for volume rendering of regular volumes. We present an iterative integration method based on the adaptive Simpson's rule and discuss different strategies to use adaptive step sizes for both the inner and the outer integrals of the volume rendering equation (1). The control of the step size for both integrals is done in consonance. We present the results of computational experiments with different datasets designed to analyze the accuracy and the efficiency of our method. We demonstrate that our method delivers accuracy control while preserving competitive performance.

The text is organized as follows: Section II reviews previous works on volume rendering related to our proposal. Section III details our adaptive method and discusses strategies to adapt the integration step. Section IV presents the results on the accuracy and the efficiency achieved by our method. Section V contains final remarks and opportunities for future work.

II. Related work

Miranda and Celes [4] considered rays crossing hexahedral cells and used Gauss–Legendre quadrature for volume rendering of unstructured meshes, evaluating the contribution of each cell along the ray analytically. Hajjar et al. [5] proposed a parameterized evaluation based on Simpson's rule, using a number of iterations to determine how parameterized intervals

are to be split. Their method is very accurate, but gets slower as the number of interactions increases.

Novins and Arvo [6] presented three adaptive integration methods, two of them based on Newton–Cotes formulas of order 1 and 2, and one based on power series expansion. They used the remainder term to control the error and the speed of the computation, by successively calculating the next step based on an estimative of the derivate at each position along the ray. However, their methods require a pre-classification of the data, to be able to estimate the derivate at any point, resulting in artifacts and loss of details. Moreover, the estimation of global error and step size does not guarantee high accuracy.

Some works focus on the accuracy delivered by numerical methods. Etiene et al. [7] provided a theoretical discussion about the discretization errors caused by the approximations made in traditional volume rendering systems, focusing on Riemann summation for both the inner and the outer integrals. Etiene et al. [8] later extended that discussion, evaluating the convergence rate of other numerical integration techniques. Hajjar et al. [5] also presented an initial discussion on numerical integration methods of order 0, 1, and 2, comparing their accuracy using adaptive and uniform sampling.

Some works propose adaptive sampling strategies, trying to ensure more samples in regions where the scalar field or the transfer function vary sharply. Lindholm et al. [9] proposed a coherent sampling geometry-based technique using the depth buffer: the number of samples is increased as the integration approaches the isosurfaces of interest. Hajjar et al. [5] argued that, using Simpson's rule for hexahedron meshes, the best way to evaluate each voxel was to get a sample on the boundaries and another at the midpoint. Because of the trilinear variation in hexahedron meshes, the scalar field can be simplified by two quadratic functions. However, their implementation was based on uniform steps because they use a 3D pre-integration table. Marchesin and de Verdiere [10] proposed a semi-analytical adaptive sampling using AMR data.

A common strategy is to use pre-integration tables to store pre-computed integration along intervals [11]. This results in an increase of performance, but demands much additional memory. Röttger et al. [12] first presented a pre-integration table, accessed via the scalar values at front and back faces of each tetrahedral cell crossed by the ray. One main drawback of using pre-integration tables is that they need to be rebuilt whenever the transfer function changes. This need was overcome by Moreland and Angel [13] using reparameterization. Hajjar et al. [5] presented two pre-integration tables using Simpson's rule: a 4D table for adaptive sampling, accessed via three scalar values and the size of the interval, and a 3D table for uniform sampling, using just the three scalar values.

III. Our integration method

We propose an adaptive integration method based on the well-known adaptive Simpson's rule. Instead of the usual recursive approach, we employ an iterative procedure: the integration adaptively evolves step by step, always taking a step forward, similar to the numerical methods for solving ordinary differential equations. This allows us to advance both inner and outer integrations in consonance without excessive re-computations.

Simpson's rule

The well-know Simpson's rule approximates the integral of a function f in an interval $[t, t+h]$ by combining three function evaluations:

$$\int_t^{t+h} f(x)\,dx \;\approx\; S(t,h) = \frac{h}{6}\left[f(t) + 4f(t+\frac{h}{2}) + f(t+h)\right]$$

The adaptive Simpson's rule, proposed by Kuncir [14] and McKeeman [15] and analysed by Lyness [16], recursively subdivides the interval of integration applying Simpson's rule until the approximation error is within a user-specified tolerance ε. The recursion is governed by the following error estimate. Using two half steps in Simpson's rule results in:

$$\int_t^{t+h} f(x)\,dx = D(t,h) + E(t,h)$$

where

$$D(t,h) = S(t,\frac{h}{2}) + S(t+\frac{h}{2},\frac{h}{2})$$

The error is estimated by comparing this with the value of Simpson's rule for a full step as follows:

$$E(t,h) = \frac{1}{15}\,|D(t,h) - S(t,h)|$$

If the estimated error exceeds the user tolerance ε, then the procedure is repeated recursively for the two subintervals $[t, t+\frac{h}{2}]$ and $[t+\frac{h}{2}, t+h]$. The function evaluations required for Simpson's rule are shared by adjacent intervals and sub-intervals, and can be cached and reused to gain performance, resulting in one function evaluation per visited interval. This is especially important for the volume rendering integral because the integrand contains an integral.

A simplified pseudo-code of Simpson's adaptive method is shown in Algorithm 1, using the notation introduced above. Note that the tolerance ε is halved together with the step size h when recursion is required. This ensures that the overall error will respect the original specified tolerance.

Algorithm 1 – Recursive adaptive Simpson's rule

input: t, h, ε
output: $\int(t,h)$
  if $E(t,h) \le \varepsilon$ **then**
    return $D(t,h) + E(t,h)$
  else
    return $\int(t, \frac{h}{2}, \frac{\varepsilon}{2}) + \int(t+\frac{h}{2}, \frac{h}{2}, \frac{\varepsilon}{2})$
  end if

Our iterative integration method

The volume rendering integral (1) is composed of two integrals: the *inner* one and the *outer* one. To keep track of the integration error, both integrals have to be considered in the adaptive approach. Although the error tolerance could be set separately for the two integrals, we have opted to use the same tolerance for both integrals in this work.

A naive adaptive procedure for computing the volume rendering integral is straightforward: Subdivide the ray length into major intervals and, for each major interval, try to compute the outer integral, subdividing the interval according to the adaptive Simpson's rule. The main problem with this naive solution is the need to also adaptively integrate the inner integral: each function evaluation for the outer integral requires the evaluation of the inner integral. This would demand a lot of re-work because it is not possible to cache all intermediate evaluations of the inner integral. Moreover, this recursive procedure cannot be mapped efficiently to GPU architectures.

To overcome these problems, we propose an iterative integration method with separate, but in consonance, step size control for both the inner and the outer integrals. Our approach is inspired in the adaptive strategies used to solve ordinary differential equations (e.g., adaptive Runge–Kutta).

Algorithm 2 shows the pseudocode of our iterative integration method. The goal is to perform the integration from t to $t+h$. The procedure also receives as input the initial step h_0. We use the adaptive Simpson's rule to evaluate the integral and the corresponding error. If the error exceeds the tolerance, we halve the step size (and the tolerance) and try to advance from t to $t+\frac{h}{2}$. If the error is within the tolerance, we accumulate the computed integral value and advance the integration to $t_{new} = t+h$. In the next iteration, the interval is from t_{new} to $t_{new}+h_{new}$. The new step h_{new} is double the current step if this is at least the second consecutive step where Simpson's rule succeeds; otherwise, the step remains unchanged.

Figure 1 illustrates how the step is adapted: dashed lines indicate failed tries; solid lines indicates succeeded tries. The numbers indicate the order of evaluation of each try. In the example shown, both the recursive and the iterative algorithms would process the tries in the same order. The recursive algorithm favors reuse of function evaluations, which are stored in the recursion call stack; the price paid is the amount of memory used. More importantly, because it stores this execution context, the recursive approach cannot be re-initialized after a succeeded step. The ability to re-initialize the integration is of crucial importance in our proposal for controlling both inner and outer step size accordingly.

Our integration method in volume rendering

We use the iterative integration method shown in Algorithm 2 as the basis for computing the VRI. In practice, we adopt two additional parameters to limit the variation of the step size: h_{min} and h_{max}. If the minimum value is reached, the step is taken even if the error is above the tolerance, when the step would normally be halved. On the other hand, the step size is never set to a value above the maximum value. The minimum value is

Algorithm 2 – Our iterative integration method

input: t, h, h_0, ε
output: $\int(t,h)$
  $I = 0$
  $t_{final} = t+h$
  $h = \min(h_0, t_{final}-t)$
  lastsuceeded = true
  while $t < t_{final}$ **do**
    if $E(t,h) \leq \varepsilon$ **then**
      $I = I + D(t,h) + E(t,h)$
      $t = t+h$
      if *lastsuceeded* **then**
        $h = 2h$ {increase step}
        $\varepsilon = 2\varepsilon$
      else
        lastsuceeded = true
      end if
      $h = \min(h, t_{final}-t)$
    else
      $h = h/2$ {decrease step}
      $\varepsilon = \varepsilon/2$
      lastsuceeded = false
    end if
  end while
  return I

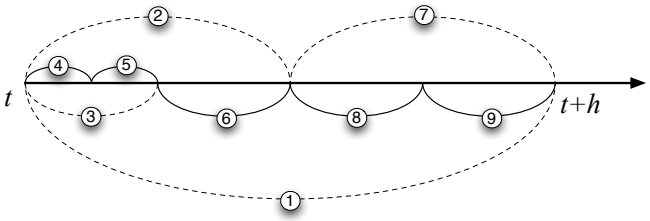


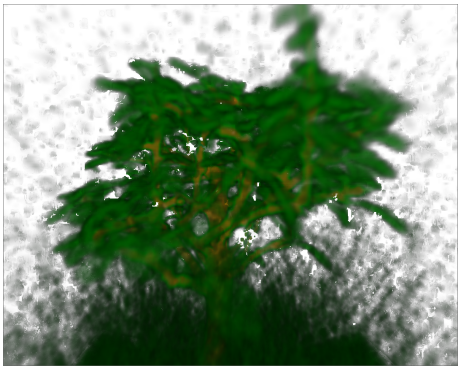
Fig. 1. Adaptive step size control.

used to avoid unnecessary reduction when, in practice, the error cannot be estimated precisely. The maximum value is used to prevent missing high function variations. Figure 2 illustrates the importance of limiting the step size to maximum value. Note the difference in the Bonsai leaves; when no maximum limit is set, some leaves are missing.

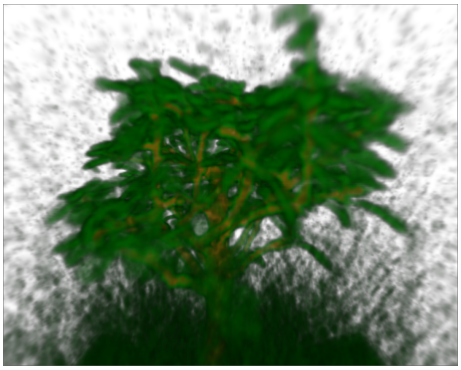
For computing the VRI using adaptive integration, we first use the inner integral to evaluate a step size, ensuring that its computation is accurate within the given tolerance. This procedure $\int_I$ is essentially the one illustrated in Algorithm 2, with the following minor changes:

- Ensure that h stays between h_{min} and h_{max}.
- Terminate as soon as a valid step size is found.
- Return the last step size along with the integral value.

The integral value returned by $\int_I$ is only used to avoid unnecessary evaluation of the outer integral when $\int_I$ is close to zero: in that case, the medium is almost transparent, with low contribution to the final image. In our experiments, we only skip the outer integral on an interval if the inner value is zero.



(a) With no h_{max} limit



(b) With $h_{max} = 8$ voxel units

Fig. 2. Importance of limiting the maximum step size.

The important value returned by $\int_I$ is the step size. The integration procedure ensures that, in this interval, the inner integral can be precisely evaluated (within the specified tolerance). We then evaluate the outer integral $\int_O$ using another variation of Algorithm 2 with these minor changes:

- Ensure that h stays between h_{min} and h_{max}.
- Perform the integration for all color channels.
- Iterate until the whole ray is covered.
- Return the last step size along with the integral value.

Inside $\int_O$, we use regular (i.e., non-adaptive) Simpson's rule to evaluate the inner integral whenever necessary, since the error is already bounded.

The pseudocode of our adaptive VRI evaluation is shown in Algorithm 3. The VRI is evaluated for all four channels in tandem. For clarity, the pseudocode does not track the tolerance value. In practice, the tolerance is adjusted accordingly as the step size is decreased or increased, as in Algorithm 2.

Coupling strategies

One important question in the VRI adaptive computation is how to couple the control of the steps used to evaluate the inner and the outer integrals. As indicated in Algorithm 3, we tried two different strategies, named *coupled* and *decoupled*.

In the coupled strategy, the inner integral dictates the initial step size to be used at each time the outer integral has to be evaluated. This is based on the assumption that both integrals are coherent, and we try to cover the outer interval with just one shot at first. Moreover, should we start the outer integration with the same step as the last evaluation of the inner integration, the intermediate values of Simpson's rule for the inner integral are already computed and can be reused.

Algorithm 3 – Proposed adaptive VRI evaluation

Input: t, h, h_0, ε
Output: VRI (t, h)

```
I = [R = 0, G = 0, B = 0, A = 0]
t_final = t + h
h_inner = min(h_0, t_final − t)
h_outer = h_inner            {only for the "decoupled" strategy}
while t < t_final do
  I_inner, h_inner = ∫_I(t, t_final − t, h_inner)
  if I_inner ≠ 0 then
    h_outer = h_inner        {only for the "coupled" strategy}
    I_outer, h_outer = ∫_O(t, h_inner, h_outer)
    I = I + I_outer
  end if
end while
return I
```

In the decoupled strategy, the outer step is controlled independently, at its own pace. The inner step is only used to limit the integration interval. In theory, the decoupled strategy seems more appropriate since it makes no assumptions on coherence. That would be the case, for instance, when the transfer function presents high frequency variation on the color channels. In practice, as we shall demonstrate, both strategies are equivalent: the coupled strategy is slightly more efficient, while the decoupled is slightly more accurate.

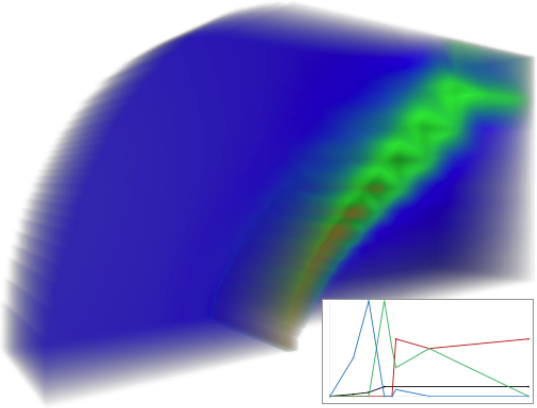
Ideally, changing the initial step size of the evaluation would only interfere in the performance of the computation. As the error is controlled, any initial step should work. However, the evaluation of the error is also based on a sampling strategy, and high frequency regions may be wrongly ignored.

IV. Results

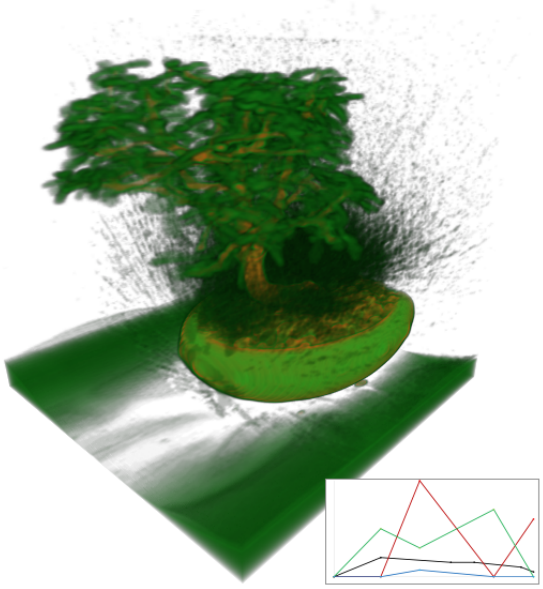
To test the accuracy and the efficiency of our method, we ran a set of computational experiments, using different datasets obtained from volvis.org [17]. The experiments were run on a computer equipped with a 2.66 Ghz Quad Core and a GeForce GTX 560. Figure 3 shows the results achieved by our approach with a prescribed error tolerance set to $\varepsilon = 0.01$ for four different datasets. In all tests, the image is 800×600 pixels.

We tested both CPU and GPU implementations of a ray casting algorithm to perform volume visualization using our proposed approach. The GPU implementation uses a GLSL fragment shader, with single precision floating point for better performance. The CPU implementation uses double precision.

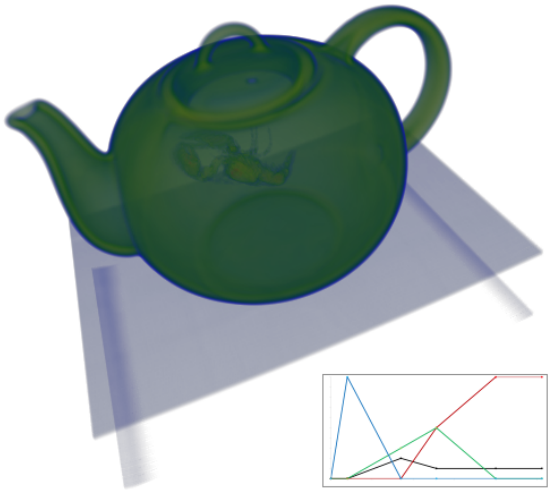
We compared our method, using both the coupled and the decoupled strategies, against the uniform sampling approach using Riemann summation [7], with different sampling distances: 0.5, 0.4, 0.3, 0.2, and 0.1 voxel unit. We analyzed accuracy by comparing the achieved result against a quality-reference image obtained using Riemann summation with a very small sampling distance ($h = 0.01$ voxel unit).



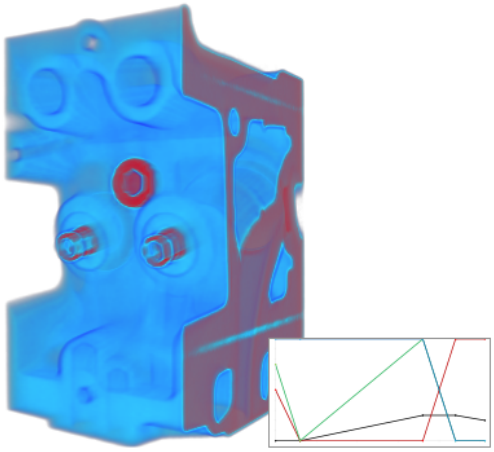
(a) BluntFin



(b) Bonsai



(c) Boston teapot

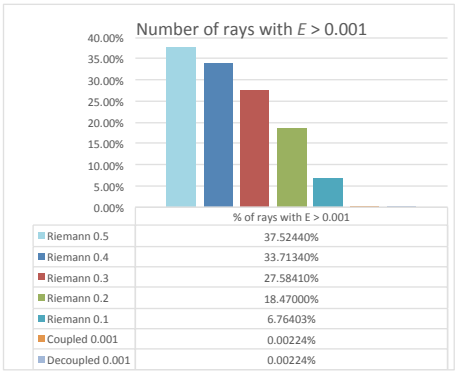


(d) Engine

Fig. 3. Results achieved by our adaptive method, with corresponding transfer functions.

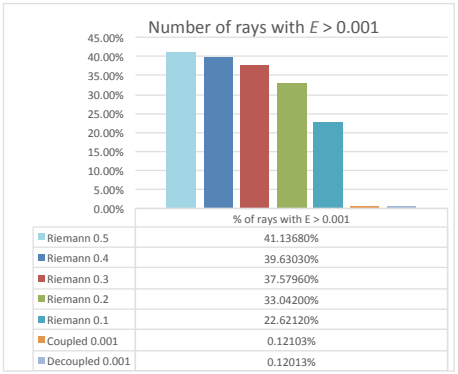
We tested different parameter configurations. Different models work better with different parameters configurations. To use a single configuration applied to the four tested models, we ran the experiments using $h_0 = 0.5$, $h_{min} = 0.1$, $h_{max} = 2.0$, all expressed in voxel units. Among the four tested models, we noticed that the Bonsai model requires a small step to ensure accuracy, while the BluntFin model limits the maximum step size because of the large blue region with low transparency. We then detail accuracy results achieved with this two models; for the other two, the algorithms presented similar results.

To compare the accuracy among the algorithms, we used the reference-quality image to compute the differences in intensity for each color channel (R, G, B, and A) in the final image. Using the CPU implementation, we ran the experiments using three different error tolerances for our algorithm: $\varepsilon = 0.01,\ 0.005,$ and 0.001. We then counted the number of rays in the final image for which the error exceeded the tolerance. In principle, these numbers should be zero but, as we have mentioned, numerical error evaluation has inaccuracies by itself and we limit the step to h_{min}. Nevertheless, our algorithm produced quite accurate results overall. For comparison, we also counted the number of rays above the same limit for the different sampling distances of Riemann summation. As expected, the error decreases as one decreases the step size. However, it is not possible to reach accurate results without using prohibitive computational costs while employing constant step sizes. The results are summarized in Table I for the Bonsai model and in Table II for the BluntFin model. Figures 4 and 5 show these results in histograms for better comparison, considering the Bonsai and BluntFin models with tolerance set to 0.001.



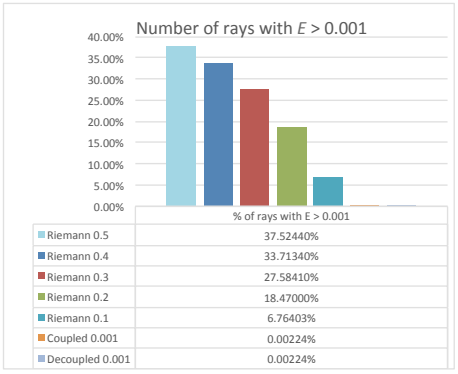
	% of rays with E > 0.001
Riemann 0.5	37.52440%
Riemann 0.4	33.71340%
Riemann 0.3	27.58410%
Riemann 0.2	18.47000%
Riemann 0.1	6.76403%
Coupled 0.001	0.00224%
Decoupled 0.001	0.00224%

(a) Red channel



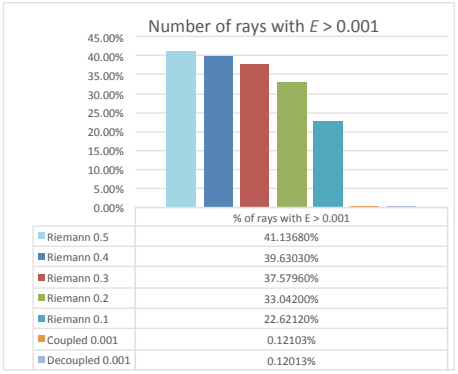
	% of rays with E > 0.001
Riemann 0.5	41.13680%
Riemann 0.4	39.63030%
Riemann 0.3	37.57960%
Riemann 0.2	33.04200%
Riemann 0.1	22.62120%
Coupled 0.001	0.12103%
Decoupled 0.001	0.12013%

(b) Green channel



	% of rays with E > 0.001
Riemann 0.5	37.52440%
Riemann 0.4	33.71340%
Riemann 0.3	27.58410%
Riemann 0.2	18.47000%
Riemann 0.1	6.76403%
Coupled 0.001	0.00224%
Decoupled 0.001	0.00224%

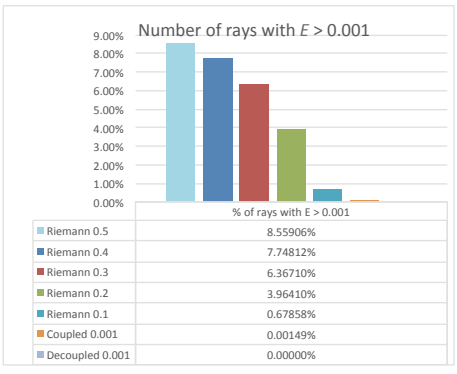
(c) Blue channel



	% of rays with E > 0.001
Riemann 0.5	41.13680%
Riemann 0.4	39.63030%
Riemann 0.3	37.57960%
Riemann 0.2	33.04200%
Riemann 0.1	22.62120%
Coupled 0.001	0.12103%
Decoupled 0.001	0.12013%

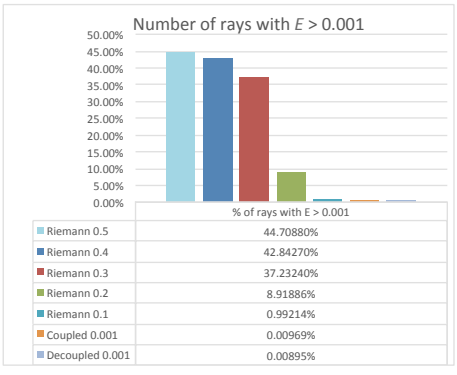
(d) Alpha channel

Fig. 4. Histograms for the Bonsai dataset with $\varepsilon = 0.001$.



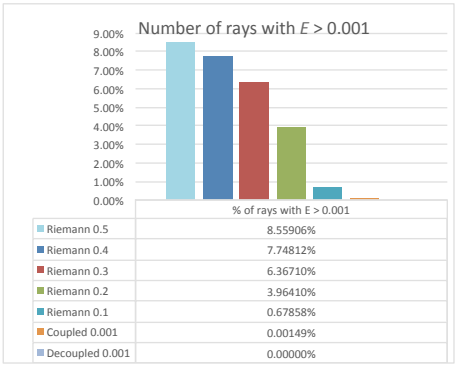
	% of rays with E > 0.001
Riemann 0.5	8.55906%
Riemann 0.4	7.74812%
Riemann 0.3	6.36710%
Riemann 0.2	3.96410%
Riemann 0.1	0.67858%
Coupled 0.001	0.00149%
Decoupled 0.001	0.00000%

(a) Red channel



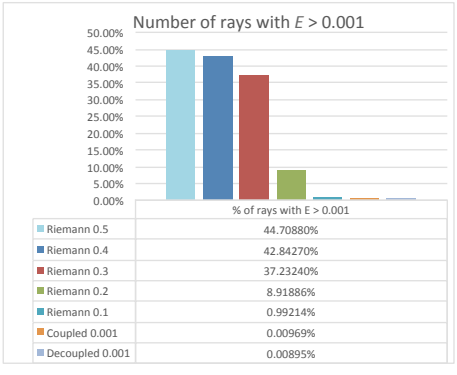
	% of rays with E > 0.001
Riemann 0.5	44.70880%
Riemann 0.4	42.84270%
Riemann 0.3	37.23240%
Riemann 0.2	8.91886%
Riemann 0.1	0.99214%
Coupled 0.001	0.00969%
Decoupled 0.001	0.00895%

(b) Green channel



	% of rays with E > 0.001
Riemann 0.5	8.55906%
Riemann 0.4	7.74812%
Riemann 0.3	6.36710%
Riemann 0.2	3.96410%
Riemann 0.1	0.67858%
Coupled 0.001	0.00149%
Decoupled 0.001	0.00000%

(c) Blue channel



	% of rays with E > 0.001
Riemann 0.5	44.70880%
Riemann 0.4	42.84270%
Riemann 0.3	37.23240%
Riemann 0.2	8.91886%
Riemann 0.1	0.99214%
Coupled 0.001	0.00969%
Decoupled 0.001	0.00895%

(d) Alpha channel

Fig. 5. Histograms for the BluntFin dataset with $\varepsilon = 0.001$.

TABLE I
PERCENTAGE (%) OF RAYS THAT EXCEEDED THE TOLERANCE ERROR FOR THE BONSAI DATASET.

	$E > 0.01$				$E > 0.005$				$E > 0.001$			
	R	G	B	A	R	G	B	A	R	G	B	A
Riemann ($h = 0.5$)	0.1139	1.8150	0.0000	12.7660	2.3645	7.6170	0.0000	23.8579	12.4298	37.5244	0.0103	41.1368
Riemann ($h = 0.4$)	0.0175	0.4904	0.0000	9.4858	0.9548	5.4722	0.0000	20.4068	11.2178	33.7134	0.0063	39.6303
Riemann ($h = 0.3$)	0.0072	0.0184	0.0000	7.1827	0.0883	2.9643	0.0000	15.4645	9.4006	27.5841	0.0036	37.5796
Riemann ($h = 0.2$)	0.0027	0.0000	0.0000	3.0763	0.0121	0.4016	0.0000	9.2177	6.2181	18.4700	0.0000	33.0420
Riemann ($h = 0.1$)	0.0000	0.0000	0.0000	0.0000	0.0004	0.0000	0.0000	2.3614	0.8328	6.7640	0.0000	22.6212
Coupled	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0022	0.0000	0.1210
Decoupled	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0022	0.0000	0.1201

TABLE II
PERCENTAGE (%) OF RAYS THAT EXCEEDED THE TOLERANCE ERROR FOR THE BLUNTFIN DATASET.

	$E > 0.01$				$E > 0.005$				$E > 0.001$			
	R	G	B	A	R	G	B	A	R	G	B	A
Riemann ($h = 0.5$)	0.2778	0.6842	0.1819	0.0720	0.7353	2.2449	1.0645	1.2796	2.8620	8.5591	43.3564	44.7088
Riemann ($h = 0.4$)	0.1670	0.2472	0.0235	0.0007	0.5306	1.2688	0.4083	0.5261	2.2554	7.7481	40.7159	42.8427
Riemann ($h = 0.3$)	0.0179	0.0559	0.0000	0.0000	0.2625	0.4228	0.0477	0.0910	1.5581	6.3671	28.4687	37.2324
Riemann ($h = 0.2$)	0.0000	0.0007	0.0000	0.0000	0.0436	0.0261	0.0000	0.0000	1.0011	3.9641	4.0547	8.9189
Riemann ($h = 0.1$)	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.2640	0.6786	0.0738	0.9921
Coupled	0.0000	0.0000	0.0000	0.0000	0.0000	0.0007	0.0000	0.0000	0.0000	0.0015	0.0917	0.0097
Decoupled	0.0000	0.0000	0.0000	0.0000	0.0000	0.0007	0.0000	0.0000	0.0000	0.0000	0.0910	0.0089

TABLE III
CPU TIME TO RENDER A FRAME (IN SECONDS).

		BluntFin	Bonsai	Teapot	Engine
Riemann	$h = 0.5$	6.53	16.23	17.97	8.86
Riemann	$h = 0.4$	8.05	20.03	21.77	10.91
Riemann	$h = 0.3$	10.52	26.31	27.87	14.33
Riemann	$h = 0.2$	15.47	38.94	40.17	21.23
Riemann	$h = 0.1$	30.33	76.80	76.55	41.70
Coupled	$\varepsilon = 0.01$	9.05	20.03	17.22	9.36
Decoupled	$\varepsilon = 0.01$	9.22	20.14	17.25	9.45
Coupled	$\varepsilon = 0.005$	9.30	23.61	18.61	10.53
Decoupled	$\varepsilon = 0.005$	9.55	23.72	18.66	10.67
Coupled	$\varepsilon = 0.001$	10.94	38.19	23.33	15.25
Decoupled	$\varepsilon = 0.001$	11.28	38.30	23.39	15.34

TABLE IV
GPU PERFORMANCE EXPRESSED IN FPS.

		BluntFin	Bonsai	Teapot	Engine
Riemann	$h = 0.5$	603	68	149	156
Riemann	$h = 0.4$	494	55	120	126
Riemann	$h = 0.3$	390	42	91	96
Riemann	$h = 0.2$	270	28	62	65
Riemann	$h = 0.1$	143	14	32	34
Coupled	$\varepsilon = 0.01$	415	31	83	81
Decoupled	$\varepsilon = 0.01$	383	31	80	76
Coupled	$\varepsilon = 0.005$	367	23	68	63
Decoupled	$\varepsilon = 0.005$	335	22	66	59
Coupled	$\varepsilon = 0.001$	226	9	34	29
Decoupled	$\varepsilon = 0.001$	205	9	33	28

For a performance analysis, we ran experiments with both the CPU and the GPU implementations. Table III reports the average time for rendering a frame on the CPU for several frames at the view angle, as illustrated in Figure 3. As can be seen, our method presents overall better performance, specially if we consider the increased accuracy. In fact, the CPU architecture favors the implementation of adaptive algorithms.

The challenge resides in developing an efficient adaptive algorithm on the GPU. In contrast to the CPU, the SIMD GPU architecture favors the implementation of the Riemann summation: all threads perform exactly the same computations. In the adaptive approach, different threads (rays) need different step sizes, making it difficult to take full advantage of the GPU parallel processing power. Table IV shows the achieved performance expressed in frames per second. As can be seen, the Riemann summation algorithms present better performance.

We then ran an additional experiment: we repeated the experiment for the Bonsai dataset with $\varepsilon = 0.001$, but now considering $h_{min} = 0.4$, and $h_{max} = 4.0$. Since each step in Simpson's rule subdivides the interval at four equidistant samples, we still expect to have better accuracy than Riemann summation with $h = 0.1$, while increasing the performance of our method, specially for the GPU implementation. In fact, as can be seen in Table V, this parameter configuration made our adaptive method competitive in terms of performance, while still delivering better accuracy, as shown in Table VI.

To illustrate the variation of step size along the rays, we tracked a ray while rendering the Bonsai model, setting $\varepsilon = 0.01$, $h_{min} = 0.4$, and $h_{max} = 4.0$. Figure 6 plots the evaluation of the alpha channel showing the samples taken by our adaptive method. As expected, the method takes larger steps in regions of low variation and smaller in regions of high variation.

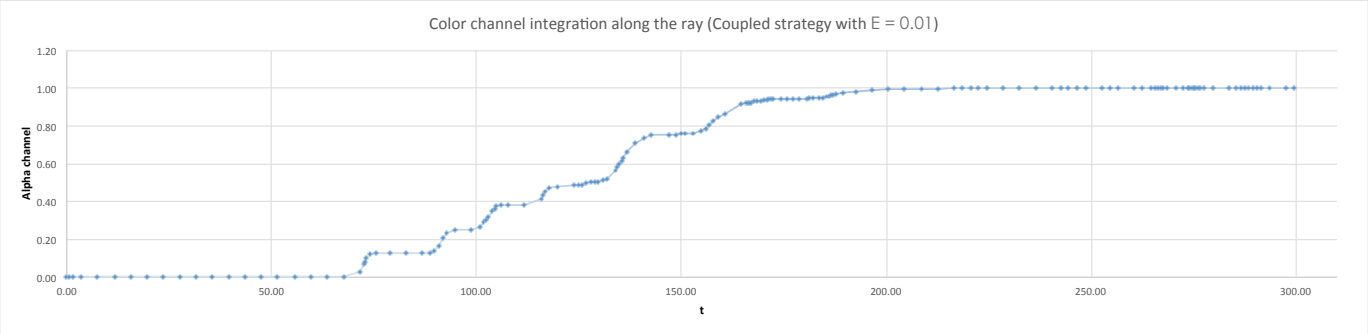


Fig. 6. Adaptive steps for the alpha channel evaluation of a ray in the Bonsai dataset.

TABLE V
CPU AND GPU PERFORMANCE FOR THE BONSAI DATASET USING $h_{min} = 0.4$ AND $h_{max} = 4.0$.

		CPU (s)	GPU (fps)
Riemann	$h = 0.1$	76.80	14
Coupled	$\varepsilon = 0.001$	26.77	19
Decoupled	$\varepsilon = 0.001$	26.91	18

TABLE VI
PERCENTAGE (%) OF RAYS THAT EXCEEDED THE TOLERANCE ERROR FOR THE BONSAI DATASET WITH $\varepsilon = 0.001$, $h_{min} = 0.4$, AND $h_{max} = 4.0$.

	$E > 0.001$			
	R	G	B	A
Riemann ($h = 0.1$)	0.8328	6.7640	0.0000	22.6212
Coupled	0.0022	0.0453	0.0000	0.4738
Decoupled	0.0027	0.0430	0.0000	0.4675

V. CONCLUSION

In this paper, we considered the evaluation of the volume rendering integral. We argued that an adaptive procedure is needed to keep overall error under a predefined tolerance. We proposed an iterative method to evaluate the volume rendering integral using the adaptive Simpson's rule. We also discussed and presented strategies to control the step size of both the internal and the external integrals in consonance.

Our computational experiments have shown that the proposed approach works as desired, maintaining accuracy under control. Regarding performance, our CPU implementation proved to run faster than Riemann summation strategies, while delivering much higher accuracy. The GPU implementation, on the other hand, performs worse than Riemann summation in general, but we have demonstrated that it can be tuned to be competitive, while presenting better accuracy. In fact, as expected, the proposed adaptive procedure allows us to balance accuracy and efficiency automatically by parameter tuning.

We are currently working on extending our method to unstructured meshes. As future work, it would be interesting to apply perceptual principles to better control the quality of the rendered images. How does the internal error tolerance affect the final image? How to better evaluate the perceptual error? Finally, it would be interesting to incorporate other illumination models, such as ambient occlusion, in the error estimation.

ACKNOWLEDGMENT

This work is part of the first author's M.Sc. work at PUC-Rio. The authors are partially supported by CNPq research grants.

REFERENCES

[1] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *Proceedings of SIGGRAPH '88*. ACM, 1988, pp. 65–74.
[2] N. Max, "Optical models for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 1, no. 2, pp. 99–108, 1995.
[3] K. Engel, M. Kraus, and T. Ertl, "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading," in *Proceedings of Workshop on Graphics Hardware*. ACM, 2001, pp. 9–16.
[4] F. Miranda and W. Celes, "Volume rendering of unstructured hexahedral meshes," *The Visual Computer*, vol. 28, no. 10, pp. 1005–1014, 2012.
[5] J.-F. El Hajjar, S. Marchesin, J.-M. Dischler, and C. Mongenet, "Second order pre-integrated volume rendering," in *PacificVIS '08*. IEEE, 2008, pp. 9–16.
[6] K. Novins and J. Arvo, "Controlled precision volume integration," in *Proceedings of Workshop on Volume Visualization*. ACM, 1992, pp. 83–89.
[7] T. Etiene, D. Jonsson, T. Ropinski, C. Scheidegger, J. Comba, L. Nonato, R. Kirby, A. Ynnerman, and C. Silva, "Verifying volume rendering using discretization error analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 1, pp. 140–154, 2014.
[8] T. Etiene, R. M. Kirby, and C. T. Silva, "A Study Of Discretization Errors In Volume Rendering Integral Approximations," in *EuroVis Workshop on Reproducibility, Verification, and Validation in Visualization*. The Eurographics Association, 2013.
[9] S. Lindholm, D. Jönsson, H. Knutsson, and A. Ynnerman, "Towards data centric sampling for volume rendering," in *SIGRAD 2013*, 2013.
[10] S. Marchesin and G. de Verdiere, "High-quality, semi-analytical volume rendering for amr data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1611–1618, 2009.
[11] M. Kraus and T. Ertl, "Pre-integrated volume rendering," in *The Visualization Handbook*. Academic Press, 2004, pp. 211–228.
[12] S. Röttger, M. Kraus, and T. Ertl, "Hardware-accelerated volume and isosurface rendering based on cell-projection," in *Proceedings of the Conference on Visualization '00*. IEEE, 2000, pp. 109–116.
[13] K. Moreland and E. Angel, "A fast high accuracy volume renderer for unstructured data," in *Proceedings of the 2004 IEEE Symposium on Volume Visualization and Graphics*. IEEE, 2004, pp. 9–16.
[14] G. F. Kuncir, "Algorithm 103: Simpson's rule integrator," *Commun. ACM*, vol. 5, no. 6, p. 347, 1962.
[15] W. M. McKeeman, "Algorithm 145: Adaptive numerical integration by Simpson's rule," *Commun. ACM*, vol. 5, no. 12, p. 604, 1962.
[16] J. N. Lyness, "Notes on the adaptive Simpson quadrature routine," *J. ACM*, vol. 16, no. 3, pp. 483–495, 1969.
[17] "volvis.org," http://volvis.org/, accessed: 2014-09-15.