Volume Graphics (2007)
H. - C. Hege, R. Machiraju (Editors)

Feature Enhancement using Locally Adaptive Volume Rendering

Stéphane Marchesin and Jean-Michel Dischler and Catherine Mongenet†

LSIIT – Louis Pasteur University – Strasbourg, France

Abstract

Classical direct volume rendering techniques accumulate color and opacity contributions using the standard volume rendering equation approximated by alpha blending. However, such standard rendering techniques, often also aiming at visual realism, are not always adequate for efficient data exploration, especially when large opaque areas are present in a dataset, since such areas can occlude important features and make them invisible. On the other hand, the use of highly transparent transfer functions allows viewing all the features at once, but often makes these features barely visible. In this paper we introduce a new, straightforward rendering technique called locally adaptive volume rendering, that consists in slightly modifying the traditional volume rendering equation in order to improve the visibility of the features, independently of any transfer function. Our approach is fully automatic and based only on an initial binary classification of empty areas. This classification is used to dynamically adjust the opacity of the contributions per-pixel depending on the number of non-empty contributions to that pixel. As will be shown by our comparative study with standard volume rendering, this makes our rendering method much more suitable for interactive data exploration at a low extra cost. Thereby, our method avoids feature visibility restrictions without relying on a transfer function and yet maintains a visual similarity with standard volume rendering.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image GenerationDisplay Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism;

1. Introduction

Traditional direct volume rendering techniques consist in integrating opacity and color values along a viewing ray by using a simple light transport model, called the volume rendering equation, inspired by the physics of light traversing a participating media (usually only absorption is considered, with no scattering). Using this approach, direct volume rendering allows one to display multiple values contributing to one pixel, unlike iso-surface extraction methods, for which a single value is considered. Therefore, volume rendering is widely used for the interactive exploration of 3D datasets since it allows one to display all data ”at once”. However, one major issue consists in defining a transfer function which associates opacity and color values to the data. Setting up this transfer function and in particular the opacity function is an intricate task, since it generally requires the expertise of the user and is deeply data dependent.

In order to address this issue, one can attempt to ease the generation of transfer functions, and keep the classical volume rendering equation. Numerous authors have attempted this [KMM*01, RBS05, KD98]. This can be done either by hand, which is, however, an excessively time-consuming and painstaking process, or automatically or semi-automatically, by introducing a specific data analysis process. In either case, it cannot be excluded that for some datasets wrong decisions could be made when building the transfer function. This has implications for the common volume rendering techniques :

- In the case of direct volume rendering, opaque features might occlude some other parts of the dataset for some or even in the worst case for all viewing directions. This

† e-mail: {marchesin,dischler,mongenet}@dpt-info.u-strasbg.fr

could make the visibility of the features from the dataset highly view-dependent. Conversely, setting up a transfer function with insufficient opacity might result in unintelligible pictures. Consequently, classical volume rendering techniques, especially based on the traditional volume rendering equation, might become impractical for efficient data exploration.

- In the case of additive blending technique, the produced pictures are often under or over saturated. Since this technique uses a simple additive blending equation, over saturation of pictures can happen very quickly and with only a very limited number of contributions. On the other hand, setting a low opacity for these contributions can result in under-saturated pictures which do not convey enough information to be useful and is furthermore highly dependent on the viewpoint.

In order to resolve the crucial issue of simultaneous visibility of data features in a view-independent fashion, one can resort to adaptive techniques. This solution consists in modifying the volume rendering integral in the hope of avoiding a potentially incorrect transfer function setup. By doing so, these methods are trading visual realism, for pictures providing better understanding of the data. In particular, the advent of non-photorealistic techniques has allowed a new range of visualization clues to be added to the generated pictures in particular in the context of volume exploration.

In this paper, we focus on this class of methods. We propose a new approach that consists in modifying the volume rendering integral to achieve better visibility of the internal structures in datasets. We do so by dynamically adjusting the volume rendering integral per-pixel. Our method is view-dependent and improves the resulting pictures by conveying more dataset features at once for all possible viewing directions. In addition, our approach further minimizes the need for a transfer function setup, since it requires only a binary classification of the dataset. The latter simply consists in identifying empty areas. Unlike completely non-photo realistic volume rendering techniques, our approach remains very close to traditional volume rendering, thus providing pictures close to usual visual realism. Yet it can be considered a non-photo realistic method since it does not respect the physical light transport equation. We have implemented our technique on the GPU, and show that it can thereby reach interactive performance.

In the next section, we detail related works. In section 3, we present our new volume rendering technique. We describe how it is possible to preintegrate locally adaptive volume rendering in section 4 and detail the implementation of our technique on the GPU in section 5. We present results with various datasets and compare our technique to other volume visualization methods in section 6. Finally, we give concluding remarks in section 7.

2. Related work

In recent years, many non-photorealistic volume visualization techniques have been proposed. These techniques trade picture realism for greater data understanding.

Saito *et al.* [Sai94] created the first non-photorealistic volume visualization technique. They achieve non-photorealistic previewing of volume fields using simple primitives such as points or lines. Ebert *et al.* [ER00] use non-photorealistic feature enhancement by modifying the color and transparency of the voxels. Viola *et al.* [VKG04, VKG05] introduce a technique called importance-driven volume rendering. This technique uses pre-defined priorities between structures in the dataset and then renders accordingly: low priority structures can be occluded by higher priority ones, while high priority structures cannot be occluded by lower priority ones. However, this requires *à priori* knowledge of the dataset in order to segment and prioritize the object features. Bruckner *et al.* [BGKG05] propose a model for preserving contextual information while prioritizing relevant information according to real-time adjustable parameters. Kraus [Kra05] introduces scale-invariant volume rendering. This technique integrates the data in data space instead of doing so in physical space, and thereby achieves scale invariance for volume visualization. By its design, this technique leads to the same color and opacity for structures bearing the same density but different thicknesses. Sato *et al.* [SSN98] modify the maximum intensity projection scheme by selecting the first value above a threshold on a given ray instead of the maximum value over the whole ray. Hauser *et al.* [HMBG00] mix two well-known volume visualization techniques, namely direct volume rendering and maximum intensity projection, into a joint method. The dataset is first segmented into two classes, and those classes are then rendered using either one or the other technique. Csébfalvi *et al.* [CMH*01] present a visualization technique which uses a gradient-based voxel selection, and only renders the relevant voxels. This results in efficient visualization of contours in the dataset. Mora *et al.* [ME04] explore order independent volume rendering and extend existing maximum intensity projection (MIP) [MGK99, ME05] and X-ray techniques. Sun *et al.* [SRR04] propose to reduce noise in confocal microscopy pictures using an adaptive technique.

In order to improve the quality of those techniques, preintegration has been proposed by Engel *et al.* [EKE01]. This technique increases the visual quality of volume rendering by pre-computing slabs of the volume rendering integral.

Even though locally adaptive techniques are widely used in the image processing field, such techniques have been rarely used for visualization. Among all the techniques discussed above, very few take advantage of view-dependent rendering to increase the amount of information in the picture. Those that do so usually require the user to go through

a complex data segmentation and/or transfer function setup phase. Since this setup is view-independent, it does not adapt to the current viewing conditions in order to maximize visibility of internal structures. Furthermore, the complexity of the preprocessing phase usually makes volume rendering unsuitable as a data exploration tool. In this paper, our purpose is to move that complex phase further in the visualization pipeline by dynamically and locally adjusting the rendering at runtime. We also aim at dataset exploration, and thus our algorithm should not require complex setup, and should be able to achieve interactive rendering. Finally, as opposed to Kraus [Kra05], we want to be able to distinguish objects with different thicknesses and therefore thin objects should not appear similarly to thick ones. Therefore, we introduce a new method which is halfway between classical photo-realistic volume visualization techniques and non-photorealistic volume visualization techniques. It locally adjusts the opacity values of the contributions, and thereby manages to keep a greater number of features visible at the same time. Since opacity values are computed on the fly, our method does not need any opacity transfer function setup, but only a binary thresholding of empty areas.

3. Locally adaptive volume rendering

The locally adaptive volume rendering technique we propose in this section aims at solving the visibility issues pertaining to volume rendering by dynamically adapting the opacity of the fragment contributions per pixel. Our technique only requires a preliminary thresholding of data that separates visible from invisible areas, as opposed to a full opacity transfer function setup. There are numerous ways to achieve such a classification (any kind of segmentation is suitable), for example using gradient-based voxel selection. In this paper we use a user-defined binary classification of empty areas in the dataset, which has proved sufficient for our purpose. Let us now present our locally adaptive technique in both contexts of DVR and additive volume rendering.

3.1. Locally adaptive DVR

Let $C(x,y)$ represent the final pixel color at (x,y), $s(i)$ the scalar value of the i^{th} sample along a viewing ray cast from (x,y), $c()$ the color transfer function for scalar s so that $c(s(i))$ is the color of the i^{th} sample, $\tau()$ the opacity function so that $\tau(s(i))$ is the opacity of the i^{th} sample, and L the length of the ray. The classical direct volume rendering integral is then as follows :

$$C(x,y) = \int_0^L c(s(t)) \exp(-\int_0^t \tau(s(u))du)dt \qquad (1)$$

It can be approximated by the following Riemann sum :

$$\tilde{C}(x,y) = \sum_{i=0}^{n-1} \tilde{c}(\tilde{s}(i))\tilde{\tau}(\tilde{s}(i)) \prod_{j=0}^{i-1}(1-\tilde{\tau}(\tilde{s}(j))) \qquad (2)$$

Let us now define *relevant* contributions : as shown on Figure 1, *relevant* contributions are samples that fall within segmented parts of the datasets (shown in blue on the figure). Let $\delta()$ be our binary classification function, so that $\delta(\tilde{s}(i))$

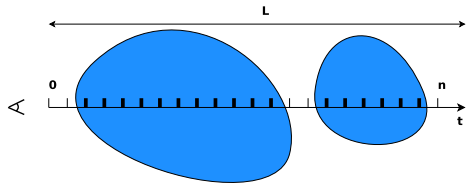

Figure 1: *Counting the relevant contributions (in bold) carried by a ray going through a segmented dataset (in blue). Among the 22 samples, there are only 17 which are relevant.*

is defined as 1 when the i^{th} sample on the ray is *relevant*, 0 otherwise. In order to keep all the structures that project to a given pixel visible at the same time, we use the same opacity $\tilde{\tau}_0$ for all the *relevant* contributions made to this pixel :

$$\tilde{C}(x,y) = \sum_{i=0}^{n-1} \delta(\tilde{s}(i))\tilde{c}(\tilde{s}(i))\tilde{\tau}_0(1-\tilde{\tau}_0)^{\sum_{j=0}^{i-1}\delta(\tilde{s}(j))-1} \qquad (3)$$

In order to keep as many features as possible visible at once, we want to compute $\tilde{\tau}_0$ that maximizes the opacity of the furthest contribution over $[0,1]$. The total opacity for the furthest contribution is then :

$$\tilde{\tau}_{furthest}(x,y) = \tilde{\tau}_0(1-\tilde{\tau}_0)^{\sum_{i=0}^{n-1}\delta(\tilde{s}(i))-1} \qquad (4)$$

To maximize this function, we take its derivative and find its zeroes :

$$\frac{d\tilde{\tau}_{furthest}(x,y)}{d\tilde{\tau}_0} = \left(1-\sum_{i=0}^{n-1}\delta(\tilde{s}(i))\tilde{\tau}_0\right)(1-\tilde{\tau}_0)^{(\sum_{i=0}^{n-1}\delta(\tilde{s}(i))-2)} \qquad (5)$$

We are only interested in opacity values in $]0,1[$ since $\tilde{\tau}_0 = 0$ and $\tilde{\tau}_0 = 1$ respectively mean fully transparent and fully opaque :

$$(1-\sum_{i=0}^{n-1}\delta(\tilde{s}(i))\tilde{\tau}_0)(1-\tilde{\tau}_0)^{(\sum_{i=0}^{n-1}\delta(\tilde{s}(i))-2)} = 0 \qquad (6)$$

$$\Rightarrow 1-\sum_{i=0}^{n-1}\delta(\tilde{s}(i))\tilde{\tau}_0 = 0 \Rightarrow \tilde{\tau}_0 = \frac{1}{\sum_{i=0}^{n-1}\delta(\tilde{s}(i))} \qquad (7)$$

Therefore, in order to maximize visibility of the furthest contributions, one should choose an opacity $\tilde{\tau}_0 = \frac{1}{\sum_{i=0}^{n-1}\delta(\tilde{s}(i))}$. We therefore have :

$$\tilde{C}(x,y) = \sum_{i=0}^{n-1} c(\tilde{s}(i))\frac{1}{\sum_{k=0}^{n-1}\delta(\tilde{s}(k))}\prod_{j=0}^{i-1}(1-\frac{1}{\sum_{k=0}^{n-1}\delta(\tilde{s}(k))}) \qquad (8)$$

3.2. Locally adaptive additive volume rendering

Let $C(x,y)$, $c()$, $\delta()$ and $\tau()$ be defined as previously. The additive blending volume rendering integral is as follows :

$$C(x,y) = \int_0^L c(s(t))\tau(s(t))dt \qquad (9)$$

which can be approximated by a Riemann sum in the following way :

$$\tilde{C}(x,y) = \sum_{i=0}^{n-1} c(\tilde{s}(i))\tilde{\tau}(\tilde{s}(i)) \tag{10}$$

In order to avoid saturating colors at a given pixel in our adaptive opacity technique, we want to average the values of the *relevant* contributions gathered over a ray :

$$\tilde{C}(x,y) = \frac{1}{\sum_{i=0}^{n-1}\delta(\tilde{s}(i))}\sum_{i=0}^{n-1} c(\tilde{s}(i))\delta(\tilde{s}(i)) \tag{11}$$

which can be rewritten as :

$$\tilde{C}(x,y) = \sum_{i=0}^{n-1} \frac{c(\tilde{s}(i))\delta(\tilde{s}(i))}{\sum_{j=0}^{n-1}\delta(\tilde{s}(j))} \tag{12}$$

3.3. Method adjustments

If the adaptive volume rendering technique is used strictly as described previously, it produces images that do not clearly depict the object structures as shown in the second image of the top row of Figure 2. In order to efficiently use this method, one has to make a few adjustments to it. The first of these adjustments bounds the opacity values. In order to achieve visual continuity, the second one filters what we call the *contribution map* : the number of *relevant* samples for each pixel. The last one adds depth cues through depth-based coloring. We now describe these adjustments in detail :

- When the opacity values are used as-is, our technique adjusts even the lowest opacities to higher values, which gives false visual clues. It occurs for instance when there are only few *relevant* contributions to a pixel, that is $\sum_{i=0}^{n-1}\delta(\tilde{s}(i))$ is small and inversely $\tilde{\tau}_0$ becomes high. We therefore decided to add an upper bound $\tilde{\tau}_{max}$ to $\tilde{\tau}_0$. This bound was determined experimentally to be within the $[0.1, 0.2]$ range. The third and fourth images of the top row of Figure 2 show $\tilde{\tau}_{max} = 0.25$ and $\tilde{\tau}_{max} = 0.12$, respectively.
- In order to avoid giving false information about the dataset, spatial continuity should be ensured in screen space. However, when too few contributions are made to a single pixel, it can result in discontinuities in the picture which look like edges. To avoid this, we filter the contribution map using a Gaussian blur. After rendering the contribution map, we run it through a blurring filter that removes most of the high-frequency data. The second pass is then done from that same map. Thanks to that improvement, the noise from the contribution map is successfully removed and internal structures become clear as shown on the bottom left of Figure 2. An alternate implementation of filtering is to render the contribution map at a lower resolution during the first pass, and stretch it using the card's native bilinear filtering capabilities. Using this functionality is faster, at the expense of less accurate results as shown by the second and third images of the bottom row of Figure 2 which show 2 times and 4 times scaling, respectively. These pictures show that it is possible to greatly improve the interactivity of our technique at the expense of visual quality.
- In the case of additive volume rendering, the depth information is lost since the blending method is commutative. This means that two contributions, one on the back of the dataset, and the other on the front, will have the same result on screen. Therefore, it is primordial to restore the depth information. In this purpose, we add depth-based coloring of the fragment samples : as the samples get further from the observer, we modulate their color proportionally to the depth of the sample. This allows an increased depth perception in the produced pictures as shown by the bottom right image of Figure 2.

4. Preintegration

Preintegration is an important feature for volume rendering. It is possible to achieve preintegration in our context of adaptive additive volume rendering. In this section, we describe how this is done. Starting from the additive adaptive volume rendering integral we have :

$$C(x,y) = \frac{\int_0^L c(s(t))\delta(s(t))dt}{\int_0^L \delta(s(t))dt} = \frac{\sum_{i=0}^{n-1}\tilde{c}(i)}{\sum_{i=0}^{n-1}\tilde{\delta}(i)} \tag{13}$$

with

$$\tilde{c}(i) = \int_{\frac{L}{n}i}^{\frac{L}{n}(i+1)} c(s(t))\delta(s(t))dt \tag{14}$$

and

$$\tilde{\delta}(i) = \int_{\frac{L}{n}i}^{\frac{L}{n}(i+1)} \delta(s(t))dt \tag{15}$$

which can be approximated as follows by assuming a linear scalar function $s()$ over the i^{th} interval :

$$\tilde{c}(i) \approx \int_{\frac{L}{n}i}^{\frac{L}{n}(i+1)} c(s(i)(1-(t-\frac{L}{n}i)) + s(i+1)(t-\frac{L}{n}i))\delta(s(i)(1-(t-\frac{L}{n}i)) + s(i+1)(t-\frac{L}{n}i))dt \tag{16}$$

and

$$\tilde{\delta}(i) \approx \int_{\frac{L}{n}i}^{\frac{L}{n}(i+1)} \delta(s(i)(1-(t-\frac{L}{n}i)) + s(i+1)(t-\frac{L}{n}i))dt \tag{17}$$

It is therefore possible to preintegrate the values of $\tilde{c}(i)$ and $\tilde{\delta}(i)$ in two separate tables and thus achieve preintegration in the context of adaptive additive volume rendering. Figure 3 shows that doing so greatly improves the quality of the pictures.

Note however that it is not convenient to preintegrate adaptive direct volume rendering since the table would have 3 entries : the two scalar values as with standard preintegration, and the current opacity. This, in turn, would require a

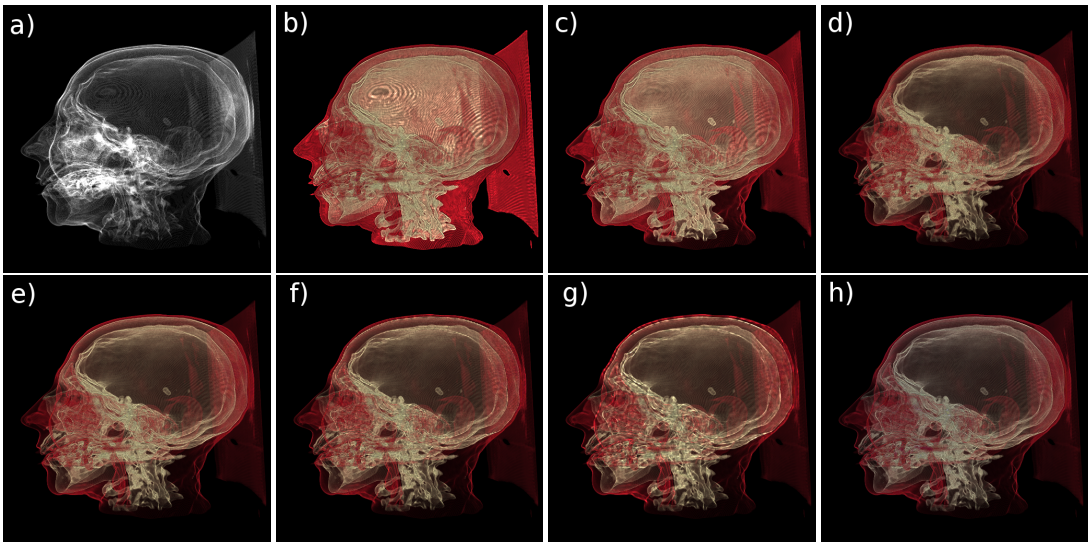

Figure 2: *(a) Contribution map (representing the number of* relevant *contributions for each pixel), Pictures produced from : (b) naive implementation with* $\tilde{\tau}_{max} = 1$ *unfiltered (20.5 fps), (c)* $\tilde{\tau}_{max} = 0.25$ *unfiltered (20.5 fps), (d)* $\tilde{\tau}_{max} = 0.12$ *unfiltered (20.5 fps), (e)* $\tilde{\tau}_{max} = 0.12$ *Gaussian filtered (8.5 fps), (f)* $\tilde{\tau}_{max} = 0.12$ *bilinear with 2 times (24.5 fps) bilinear scaling, (g) 4 times (25.3 fps) bilinear scaling and (h) z-based coloring with* $\tilde{\tau}_{max} = 0.12$ *and Gaussian filtering (8.5 fps).*

3D texture to store the preintegration table which implies a high video memory usage.

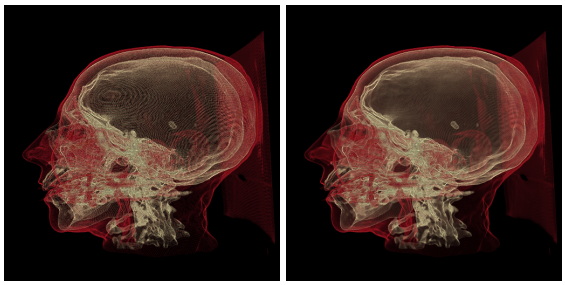

Figure 3: *Our locally adaptive volume rendering without preintegration (left) and with preintegration (right) using the same sampling rate.*

5. Implementation

We have implemented locally adaptive volume rendering on graphics hardware for voxel datasets. Our hardware implementation is essentially a 2-pass slicing-based approach using 3D textures. It is depicted in figure 4 and works as follows :

- Initially, a binary classification based on the scalar value of each voxel is computed
- In a **first pass**, the number of *relevant* contributions to each screen pixel is counted by using the binary voxel classification, thereby creating the contribution map. In order to count the contributions at each pixel, this pass does a binary rendering of the dataset into a frame buffer object : it adds 1 to the pixel if the fragment is *relevant*, and 0 otherwise.
- In a **second pass**, given the contribution map and the color transfer function, the final rendering is produced. The frame buffer object of the contribution map is bound to a texture and the dataset is rendered again. As shown in Figure 5, the fragment program renders the dataset on-screen and uses the fragment coordinate information to retrieve the number of contributions forming the current pixel (which had been computed during the first pass). This value is then used to modulate the opacity value through a look-up table (alpha_lookup), and the resulting fragment is written to the frame buffer.

6. Results

We have experimented locally adaptive volume rendering with both DVR and additive volume rendering. In all tests, the binary function used for the locally adaptive picture is based on the opacity function of the DVR rendering, and is 1 when the opacity function is greater than zero, and 0 otherwise. Figure 6 is a comparison between DVR and adaptive additive volume rendering for the $256 \times 256 \times 256$ head dataset.

These pictures show that our technique successfully maintains the internal head structures visible contrary to DVR. The DVR picture uses an opacity function that segments the bones and the skin. Figure 7 shows pictures obtained us-

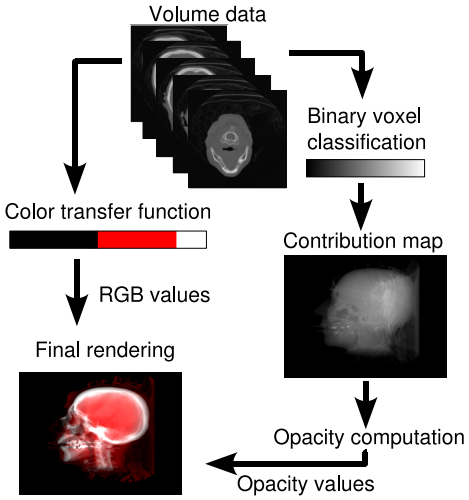

Figure 4: *Outline of our implementation.*

```
uniform sampler2D transfer_function,contributions;
uniform sampler3D dataset;
uniform sampler1D alpha_lookup;
uniform float fbo_x_size,fbo_y_size;
void main()
{
  float scalar1,scalar2,contrib,xc,yc;
  vec4 color;
  scalar1=float(texture3D(dataset, gl_TexCoord[0].xyz));
  scalar2=float(texture3D(dataset, gl_TexCoord[1].xyz));
  xc=gl_FragCoord.x/fbo_x_size;
  yc=gl_FragCoord.y/fbo_y_size;
  contrib=float(texture2D(contributions, vec2(xc,yc)));
  color=texture2D(transfer_function,vec2(scalar2,scalar1));
  gl_FragColor.rgb=color.rgb;
  if (color.a>0.0)
    gl_FragColor.a=float(texture1D(alpha_lookup, contrib));
  else
    gl_FragColor.a=0.0;
}
```

Figure 5: *Shader code for the second pass of locally adaptive additive volume rendering.*

ing our technique with different datasets. The first one is the $256 \times 256 \times 128$ bonsai dataset, and the second one is the $128 \times 128 \times 128$ bucky ball dataset. To obtain these pictures, we used a hand-tuned transfer function. As seen on these pictures, locally adaptive volume rendering can significantly enhance details. In particular, borders are made sharper in the case of DVR, while the rest of the features are still visible. In the case of additive volume rendering, locally adaptive volume rendering also avoids saturating pictures, but again keeps most details visible. This is exemplified by the bonsai dataset renderings, where the leaves are made distinguishable by the adaptive approaches. Notice that the second dataset (the bucky ball dataset) is synthetic, and does not feature clear boundaries between the different densities. Nevertheless, our technique is able to keep good

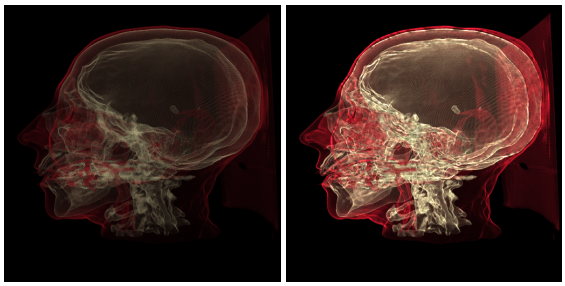

Figure 6: *Standard DVR (left), locally adaptive additive volume rendering (right). The right picture depicts more details, but did not require a full opacity transfer function setup.*

visibility of the internal structures of this dataset. Figure 8 compares locally adaptive rendering with the classical DVR technique using two opacity transfer functions and different viewpoints, for the $379 \times 229 \times 305$ knee dataset. DVR is used with two opacity functions : a naive opacity function (the opacity transfer function is 0 everywhere except in the range $[5,85]$ where it is 1) and a hand-tuned opacity function that segments the skin and bones. The locally adaptive technique uses the naive opacity function as a binary classification. These pictures show that even though it uses a naive transfer function, our technique is able to convey most of the bone structure, similarly to what is obtained with a hand-tuned transfer function and DVR. As can be seen on the left column, DVR with a naive transfer function does not allow clearly distinguishing the bone structure, whereas the adaptive approach with this same naive function makes this structure visible, in a way quite similar to the case of a hand-tuned transfer function. Furthermore, Figure 8 also shows that viewpoint changes do not impact visibility of features with our technique, as opposed to DVR.

Figure 2 shows the influence of applying a 3×3 Gaussian filter on the contribution map : small features that look like edges because of the adaptive opacity are successfully removed using this filter, while the rest of the features is still visible. Table 1 shows the performance of our technique compared to the classical volume rendering techniques (both DVR and additive volume rendering lead the same performance results) with and without preintegration. These measurements were conducted on an Athlon 4800+ machine with a GeForce 7800 GT graphics card. They show that our technique does impact the rendering interactivity only slightly.

7. Conclusions

In the context of data exploration, non-photorealistic techniques have shown that it is possible to increase the quality of the visualization by showing more data features. In this paper, we have introduced a new simple volume rendering technique that manipulates the opacity values in a

	Classical	Bilin 2x	Filter
Head	32.2 fps	29.5/24.5 fps	10.3/8.5 fps
Bonsai	55.4 fps	48.3/40.1 fps	13.5/12.4 fps
Teddy	84.2 fps	70.3/63.6 fps	22.1/21.7 fps
Knee	35.8 fps	28.4/24.8 fps	11.3/10.7 fps

Table 1: *Performance of the classical vs locally adaptive volume rendering techniques (not preintegrated/preintegrated).*

view-dependent fashion in order to ensure maximal visibility of the internal data structures. We have compared this technique to other widely used volume rendering methods, and have demonstrated its efficiency. Our technique results in better understanding of the objects features, and furthermore does not require any complex opacity function setup but only a simple binary classification of relevant voxels. Our method also ensures good visibility of the data features independently of the viewpoint, as opposed to the classical DVR method. Moreover, since our technique has been fully implemented on graphics hardware, we achieve interactive performance, thereby making it efficient in the context of data exploration, and allowing the user to use motion and interaction in order to better understand the internal structures of the dataset.

However, we think a lot of extensions are possible. First, we would like to extend the idea of locally adaptive opacity to other volume rendering algorithms, in particular when multiple techniques are used at the same time (for example, DVR and isosurfaces). Second, we would like to derive more complex opacity modification functions, for example taking the depth position of the sample into account. Finally, we would like to experiment combining our technique with an automatic segmentation technique in order to form a fully automatic volume exploration tool.

References

[BGKG05] BRUCKNER S., GRIMM S., KANITSAR A., GRÖLLER M. E.: Illustrative context-preserving volume rendering. In *Proceedings of EuroVis* (2005), pp. 69–76.

[CMH*01] CSEBFALVI B., MROZ L., HAUSER H., KÖNIG A., GRÖLLER M. E.: Fast visualization of object contours by non-photorealistic volume rendering. In *Eurographics* (2001), pp. 452–460.

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware* (2001), ACM Press, pp. 9–16.

[ER00] EBERT D., RHEINGANS P.: Volume illustration: non-photorealistic rendering of volume models. In *Proceedings of the IEEE Visualization conference* (2000), pp. 195–202.

[HMBG00] HAUSER H., MROZ L., BISCHI G.-I., GRÖLLER E.: Two-level volume rendering-fusing mip and dvr. In *Proceedings of the IEEE Visualization conference* (2000), pp. 242–252.

[KD98] KINDLMANN G. L., DURKIN J. W.: Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE Symposium On Volume Visualization* (1998), pp. 79–86.

[KMM*01] KNISS J., MCCORMICK P., MCPHERSON A., AHRENS J., PAINTER J., KEAHEY A., HANSEN C.: Interactive texture-based volume rendering for large data sets. *IEEE Computer Graphics and Applications 21*, 4 (2001), 52–61.

[Kra05] KRAUS M.: Scale-invariant volume rendering. In *Proceedings of the IEEE Visualization conference* (2005), IEEE Computer Society, pp. 295–302.

[ME04] MORA B., EBERT D. S.: Instant volumetric understanding with order-independent volume rendering. *Comput. Graph. Forum 23*, 3 (2004), 489–498.

[ME05] MORA B., EBERT D. S.: Low-complexity maximum intensity projection. *ACM Trans. Graph. 24*, 4 (2005), 1392–1416.

[MGK99] MROZ L., GRÖLLER E., KÖNIG A.: Real-time maximum intensity projection. In *Data Visualization*. 1999, pp. 135–144.

[RBS05] ROETTGER S., BAUER M., STAMMINGER M.: Spatialized transfer functions. In *Proceedings of EuroVis 2005* (2005), pp. 271–278.

[Sai94] SAITO T.: Real-time previewing for volume visualization. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization* (1994), pp. 99–106.

[SRR04] SUN Y., RAJWA B., ROBINSON J. P.: Adaptive image-processing technique and effective visualization of confocal microscopy images. In *Microscopy Research and Technique* (2004), pp. 156–163.

[SSN98] SATO Y., SHIRAGA N., NAKAJIMA S.: Local maximum intensity projection (lmip): A new rendering method for vascular visualization. *Journal of Computer Assisted Tomography, Vol. 22, No. 6, November-December 1998* (1998), 912–917.

[VKG04] VIOLA I., KANITSAR A., GRÖLLER M. E.: Importance-driven volume rendering. In *Proceedings of the IEEE Visualization conference* (2004), pp. 139–145.

[VKG05] VIOLA I., KANITSAR A., GRÖLLER M. E.: Importance-driven feature enhancement in volume visualization. *IEEE Transactions on Visualization and Computer Graphics 11*, 4 (2005), 408–418.

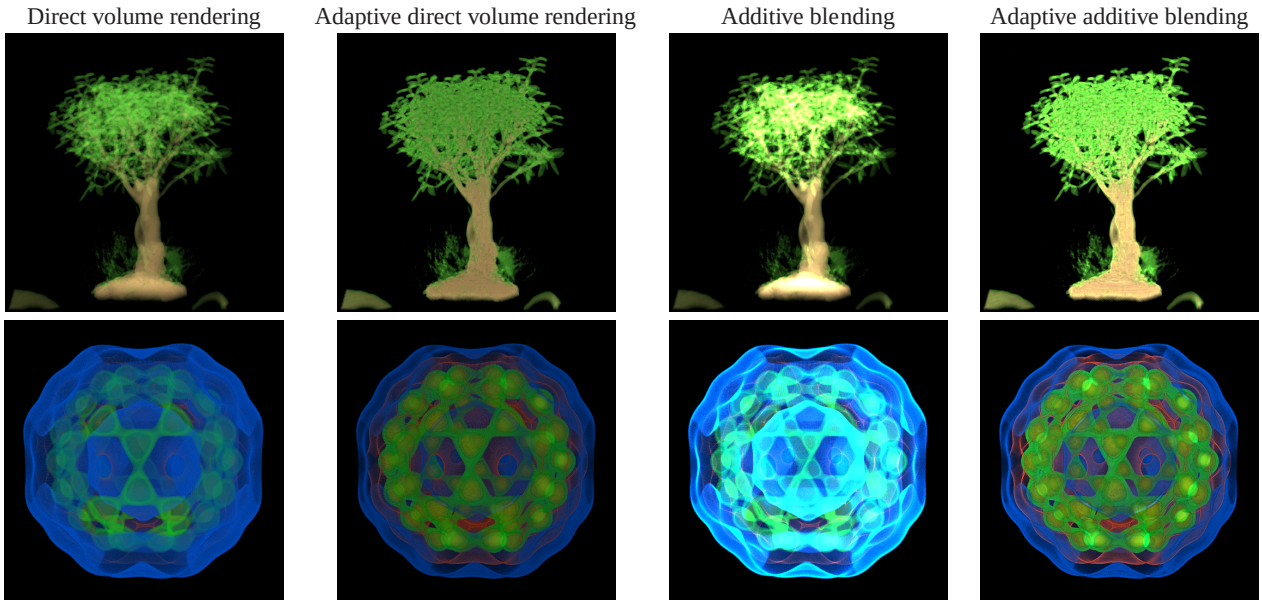

Figure 7: *Comparison between the classical and adaptive volume rendering techniques.*

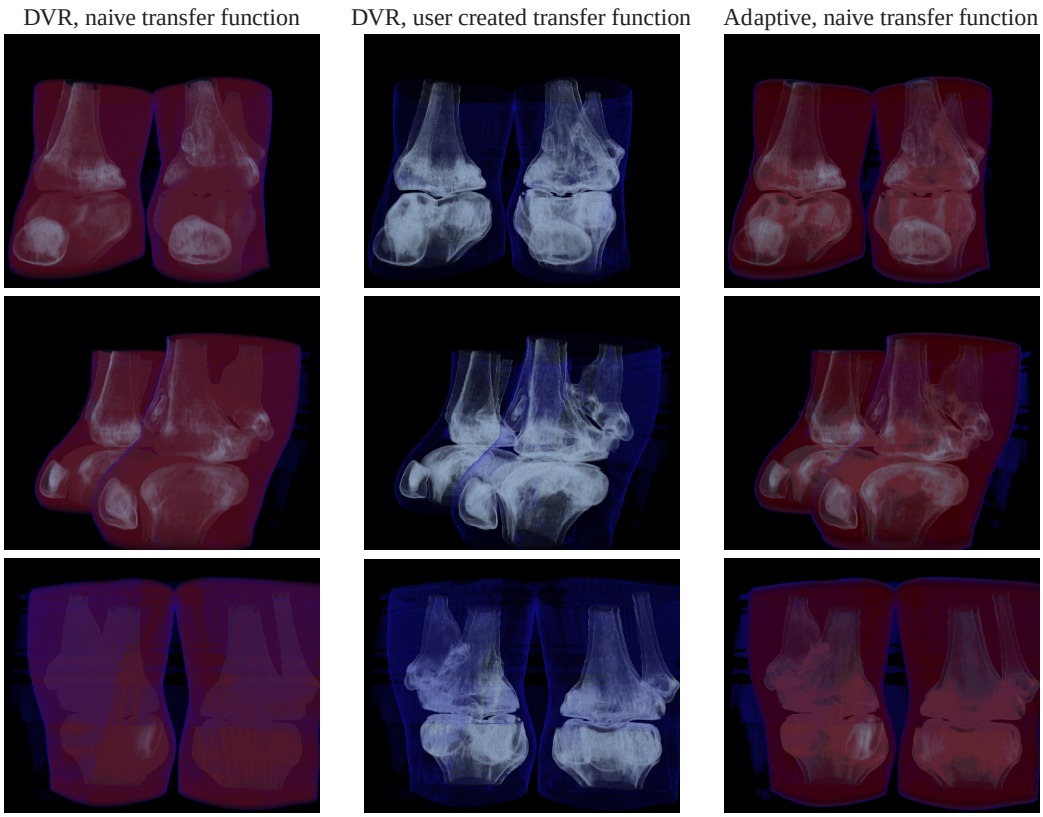

Figure 8: *Comparison between DVR (naive and hand-tuned transfer functions) and locally adaptive volume rendering (naive binary classification).*