SIGRAD 2013
T. Ropinski and J. Unger (Editors)

Poor Man's Rendering Of Segmented Data

Stefan Lindholm[1] and Alexander Bock[1]

[1]Scientific Visualization Group, Linköping University, Sweden

Abstract

In this paper we present a set of techniques for fast and efficient rendering of segmented data. Our approach utilizes the expected difference between two co-located texture lookups of a label volume, taken with different interpolation filters, as a feature boundary indicator. This allows us to achieve smooth class boundaries without needing to explicitly sample all eight neighbors in the label volume as is the case with previous methods. We also present a data encoding scheme that greatly simplifies transfer function construction.

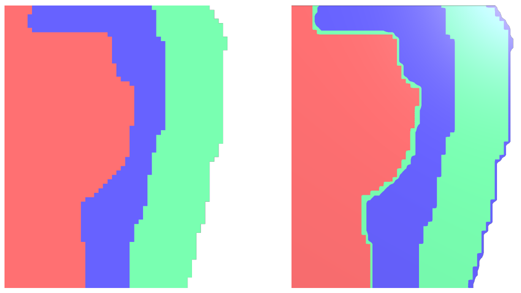
Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Picture/Image Generation—Antialiasing I.4.6 [Computer Graphics]: Segmentation—Pixel classification

1. Introduction

Rendering of segmented data is a core topic in the field of volume rendering. It is characterized in that it utilized external sources for the classification of grid points, rather than relying on user driven classification through, for example, a transfer function. By spending more resources on the classification as a pre-process step, much more reliable feature boundaries can be identified than what is possible through classification schemes applied at rendering time. Naturally, the topic of rendering segmented data is closely related to that of the actual segmentation. See [KDC*00, HJ04, UH00] for comprehensive overviews of the available literature. In this paper, we assume that a full segmentation has been performed such that the *source data* is complemented with a *label volume*, i.e. an integer volume of the same dimensionality of the source data with a single per-voxel label denoting the class membership (material) of the voxel.

The basic idea behind rendering segmented data is identical to standard volume rendering: to select visual parameters based on the class membership of each sample. This is greatly simplified since the required class membership can be accessed directly though the label volume. Therefore, the visual properties are often defined on a per-class basis, rather than in a single global definition for the entire data. The complexity of the per-class visual properties vary depending on what the situation requires and can span from a unique color to a fully defined class-specific transfer function. In this paper we use a single transfer function per class but apply the same shading scheme to all classes, however, it should be noted that this is a stylistic choice rather than a requirement of the approach.

A problem that arises when rendering segmented data is that the basic approach to 'just sample the label volume' to extract class memberships is not as straight forward as one might think. The simplest solution is, of course, to apply nearest neighbor sampling, which is guaranteed to return a valid class membership. Unfortunately, this leads to boundaries with a very blocky appearance (see Figure 1(a)). The membership operation is said to have *voxel-resolution*. The most intuitive way to achieve a higher, *pixel-resolution*, membership operation is naturally to allow for interpolation when sampling the label volume. This is, however, not guaranteed to return a valid class label unless there is no more than two classes in the entire volume. For example, an interpolation in a boundary area between class 3 and class 5 would return a class label of 4 even if this was not existent at that particular location in the data (see Figure 1(b)). An approach to achieve a pixel-resolution membership operation for segmented data, called two-level volume rendering, was presented in [HBH03]. In this approach, all eight neighboring grid points in the label volume are sampled using nearest neighbor interpolation in order to acquire the information necessary to remap the 3–5 operation to a 0–1 range and thereby avoid illegal interpolations. The result is much smoother boundary representations that are more visually pleasing. Unfortunately, the method requires an additional *seven* texture lookups per step along the ray which is many times unfeasible.



(a) Nearest neighbor (b) Linear

Figure 1: Using the two native interpolation kernels alone will result in undesired results. Nearest neighbor interpolation (a) will create a blocky result, visually unpleasing. Linear interpolation (b) will produce misclassifications between labels (see I and II) as the labels are integer values, and naïve linear interpolation will create a linear transition between the source values.

In this paper we present a novel method that delivers results comparable to pixel-resolution schemes while only requiring a *single* extra texture lookup. The core of our approach is to sample the level volume twice at the same location, with and without interpolation (i.e, with nearest neighbor vs. linear filtering). The difference between the two values is then used to smoothen the visual representation near class boundaries without creating incorrectly interpolated class values. We show that our method produces results visually comparable to methods that utilize full neighborhood sampling while requiring a factor of 7 : 1 less additional texture lookups. Additionally, we also present a data compression approach that removes the need to have the label volume accessible during rendering.

2. Poor Man's Rendering With Label Volume

The implementation will be presented in three steps. First we provide a few brief details on how to sample a single texture with different interpolation filters in OpenGL. We then present how to compute the actual class membership of each sample as well as the attenuation parameter t that is used to smoothen the boundaries.

Nearest Neighbor and Linear Sampling

As previously noted, the implementation relies on sampling the same volume twice with different interpolation filters. For many architectures, however, a single texture can only be associated with a single sampling mode. One way to get around this restriction is to always force the label volume to be associated with linear interpolation. The nearest neighbor sample can then be accessed by adding an offset that forces the sample to be taken at the center of the nearest voxel. The nearest grid point can be accessed as follows

$$s_{\text{near}} = \text{floor}\big(s + vec3(0.5)\big) \tag{1}$$

where s is the original sample position along the ray and s_{lin} is the position of the nearest neighbor. For OpenGL/GLSL both samples can then be accessed using `texture3D`. Note, that OpenGL uses voxel centric values, as opposed to grid centric values, for `texture3D` and that the sample positions needs to be mapped accordingly. Alternatively. the nearest neighbor sampling can be achieved using `texelFetch` which should not require any additional mapping of the sampling point.

Once the linear and nearest sample points have been computed, they are used to extract the linear and nearest values from the label volume, l_{lin} and l_{near} respectively.

Class Membership and Class Attenuation

In our approach the class membership is directly decided by the nearest neighbor sample in the label volume, l_{lin}. This means that membership itself is defined at voxel-resolution. To achieve smoother class boundaries we introduce a *class attenuation parameter*, t.

The attenuation parameter is computed from the nearest and linear label values

$$t = \text{abs}(l_{\text{lin}} - l_{\text{near}}). \tag{2}$$

In Equation 2, we see that $t = 0$ as long as the interpolated value is identical to the nearest neighbor value. This will be the case as long as the local neighborhood around the sample point only contains a single class. We can also note that if the local neighborhood contains two or more classes, then $t > 0$. The class attenuation parameter thereby functions as a boundary indicator.

Interpreting the attenuation parameter as a boundary indicator, we use it to adjust the opacity of the current sample. Ideally, we want full opacity when the samples are fully inside a single class and zero opacity when samples are half way in between two (or more) classes. Unfortunately, this requires us to know all classes that affect the interpolated value, which in turn would require us to sample the entire neighborhood in the label volume (which we don't want to do). Instead, we always map the opacity to zero as t reaches 0.5

$$\alpha' = \left(1 - (2t)^2\right)\alpha. \tag{3}$$

The full process is described in Algorithm 1 and illustrated in Figure 2.

While Equation 3 largely achieves the desired effect, it is important to note that it introduces a few predictable irregularities. First, $t = 0.5$ only corresponds to the half way mark between two labels if their numerical labels are separated by a single unit (e.g. 1–2, 3–2). For all other cases, the opacity will reach zero before the half way mark. Second,

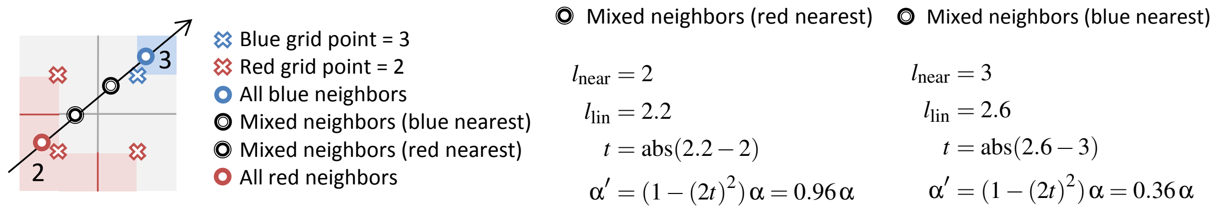

3
2
Blue grid point = 3
Red grid point = 2
All blue neighbors
Mixed neighbors (blue nearest)
Mixed neighbors (red nearest)
All red neighbors
Mixed neighbors (red nearest)
$l_{\text{near}} = 2$
$l_{\text{lin}} = 2.2$
$t = \text{abs}(2.2 - 2)$
$\alpha' = (1-(2t)^2)\,\alpha = 0.96\,\alpha$
Mixed neighbors (blue nearest)
$l_{\text{near}} = 3$
$l_{\text{lin}} = 2.6$
$t = \text{abs}(2.6 - 3)$
$\alpha' = (1-(2t)^2)\,\alpha = 0.36\,\alpha$

Figure 2: Example computations of ray contributions. The visible ray segment starts inside the red feature (class id 2), and ends inside the blue feature (class id 3). The class membership of the two samples taken in between the features needs to be decided. In our approach, their membership is fully determined by a nearest neighbor query in the label volume (here resulting in $l_{\text{near}} = 2$ and $l_{\text{near}} = 3$ respectively). However, in order to achieve smooth boundaries at pixel-resolution we employ an alpha modulation based on the difference between the the nearest query and another query using linear interpolation (here resulting in $l_{\text{near}} = 2.2$ and $l_{\text{near}} = 2.6$ respectively).

Algorithm 1 *Poor Man's Rendering **with** Label Volume*

```
Require:
  A source volume $V_{\text{src}}$ and a label volume $V_L$,
  both present on the GPU
  A set of labels $\{l_1, l_2, \cdots, l_n\}$
  A transfer function, TF$_i$ for each label $l_i$
Algorithm:
  for all sample points along ray do
    ◇ Sample $V_L$ using nearest neighbor interp. $\rightarrow l_{\text{near}}$
    ◇ Sample $V_L$ using linear interpolation $\rightarrow l_{\text{lin}}$
    ◇ Compute attenuation parameter $t$ according to Eq. 2
    ◇ Compute alpha modulation according to Eq. 3
    ◇ Sample $V_{\text{src}}$ using linear interpolation $\rightarrow v_{\text{lin}}$
    ◇ Select a TF based on $l_{\text{near}}$
    ◇ Evaluate TF$_{\text{near}}(v_{\text{lin}})$
    ◇ Apply alpha modulation to output
    ◇ Composite output to result
  end for
```

in some rare corner cases, the nearest neighbor value may change before the opacity has reached zero, effectively creating a sharper than intended cutoff. In short, we have less fine-tuned control over the opacity behavior across boundary regions.

What makes Poor Man's Rendering such an attractive trade-off is that the visual impact of the expected irregularities is near insignificant and thus acceptable given the increase in performance compared to the full neighborhood analysis.

3. Poor Man's Rendering Without Label Volume

In this section we present a data encoding scheme that eliminates the need to upload the label volume to the GPU. The encoding relies on a set of linear mappings (one for each class) from the source volume to an encoded target volume. The key here is to ensure that the co-domain of the mappings each span a unique value range in the target volume.

For example, if we have three classes which voxels exhibit overlapping value ranges (0.3–0.5, 0.2–0.7 and 0.1–0.5). We then map the value ranges of these classes to three unique value ranges in the target volume

$$l_1 : [0.3, 0.5] \mapsto [0.1, 0.3]$$
$$l_2 : [0.2, 0.7] \mapsto [0.4, 0.6]$$
$$l_3 : [0.1, 0.5] \mapsto [0.7, 0.9].$$

This mapping makes it possible to determine the class of a sample solely based on which unique range it belongs to. Note that interpolation in this volume can still lead to invalid results. The data encoding is performed as a pre-process step and the encoded volume replaces the source volume on the GPU. Meta information from the mappings also needs to be uploaded in order to decode the volume during rendering. The decoding process is simply the inverse of the linear mapping for each class.

The computation of the attenuation parameter t without a label volume works as follows. Two samples are taken from the encoded volume, with nearest neighbor (e_{near}) and linear interpolation (e_{lin}) respectively. First, the nearest neighbor sample is used to identify the label and its valid value range (from the uploaded mapping information)

$$e_{\text{near}} \rightarrow l_{\text{near}},\ [e_{\min}, e_{\max}]. \tag{4}$$

Based on this the attenuation parameter t can be computed as

$$t = \text{abs}(v_{\text{lin}} - \frac{e_{\max} + e_{\min}}{2}) - \frac{e_{\max} - e_{\min}}{2} \tag{5}$$

clamped to the 0–1 range. Poor Man's Rendering can now be performed using Equations 5 and 3. The full process is described in Algorithm 2.

A positive side effect to the encoding is that the user can now specify a single global transfer function without labels while still maintaining separate visual properties for the different classes (since the value ranges are known to be non-overlapping). This is particularly beneficial for systems that are not previously set up to load and render segmented data

Algorithm 2 *Poor Man's Rendering **without** Label Volume*

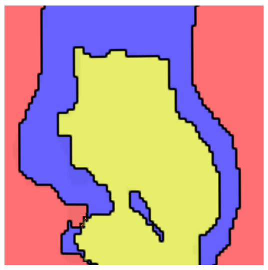
```
Require:
  A source volume $V_{\text{src}}$ and a label volume $V_L$
  A target volume for the encoding $V_{\text{enc}}$
  A set of labels $\{l_1, l_2, \cdots, l_n\}$
  A transfer function, $\text{TF}_i$ for each label $l_i$
Pre-process:
  for all labels $l_i$ do
    ◇ Find the value range $[r_{\min}, r_{\max}]$ covered by by the voxels of class $i$ in $V_{\text{src}}$
    ◇ Assign a unique range $[e_{\min}, e_{\max}]$ where $e_{\min} = 0.1 + 0.3i$ and $e_{\max} = 0.2 + 0.3i$ in $V_{\text{enc}}$
    ◇ Linearly map all voxels of class $i$ from $V_{\text{src}} \mapsto V_{\text{enc}}$ as $[r_{\min}, r_{\max}] \mapsto [e_{\min}, e_{\max}]$
    ◇ Save mapping information as $M_i$
  end for
  ◇ Upload $V_t$ to GPU together with meta information of each class mapping
Algorithm:
  for all sample points along ray do
    ◇ Sample $V_{\text{enc}}$ using nearest neighbor interp. $\rightarrow e_{\text{near}}$
    ◇ Identify current label $l_{\text{near}}$ and valid value range $[e_{\min}, e_{\max}]$ based on which unique range that contains $e_{\text{near}}$
    ◇ Sample $V_{\text{enc}}$ using linear interpolation $\rightarrow e_{\text{lin}}$
    ◇ Compute attenuation parameter $t$ according to Eq. 5
    ◇ Compute alpha modulation according to Eq. 3
    ◇ Compute the inverse mapping $e_{\text{lin}} \mapsto v_{\text{lin}}$ using $M_j$
    ◇ Evaluate $\text{TF}(v_{\text{lin}})$
    ◇ Apply alpha modulation to output
    ◇ Composite output to result
  end for
```

or multiple volumes. A potential issue with this type of encoding is the loss of precision that follows from the remapping. Since this naturally becomes a situational trade-off a complete analysis is out of the scope of this paper. What can be said is that as the more narrow the value ranges are per class in the source volume, the less precision will be lost.

3.1. Results

In this section we present the results that we achieve using our two proposed techniques. The real world dataset we use is the Walnut with its accompanying segmentation. For the comparisons we have applied alpha attenuation in boundary regions also for two-level volume rendering although this was not a part of the original presentation.

Figure 5 shows a comparison between our implementation of the two-level volume rendering to our method both in the cases where a label volume is present and in the case of the encoded volume. The test images were created using a small stack of slices of the Walnut dataset with each label value assign to a unique, fully opaque color. It is clearly visible that both interpolation kernels produce a superior visual quality when compared with the nearest neighbor interpolation. However, despite minor (expected) visual artifacts (see lower left corner in 5(c)) our method performs reasonably well and achieves a comparable result to the method employing a full neighborhood search for each segment. Figure 3 shows the two-level volume rendering and the our method applied to a single slice of the volume. Note how the background color is visible between feature boundaries as the opacity of the samples is reduced by the attenuation factor t.


(a) Two-Level Volume Rendering

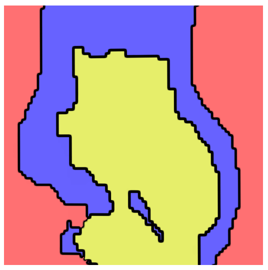
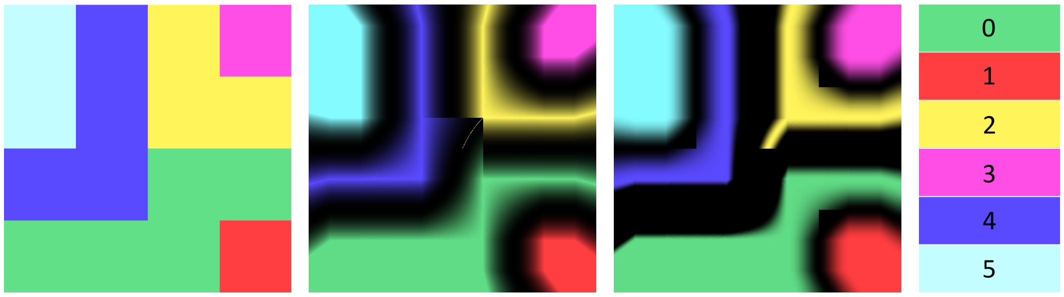

(b) Poor Man's Rendering

Figure 3: Comparing a single slice off the Walnut dataset. In the areas between labeled regions, the opacity of the single sample is lowered due to the attenuation factor t and therefore the background color is visible. Note that the thickness of the boundary region in our method is depending on the label values, whereas it is normalized in the two-level volume rendering approach. This is visible in (b) as the width between the violet-red boundary is thinner than the violet-yellow boundary.

Figure 6 is a comparison of the walnut dataset showing the different structures within the walnut. The visual result for each segment is determined by its own, separate 1-dimensional transfer function. Figure 6(a) and 6(b) are rendered with the same rendering parameters and the differences between the two techniques are shown in Figures 6(c) and 6(d) with the difference magnified by a factor of 10. As expected, the bulk of difference is in the region around the center, where a segment with label value 4 is adjacent to the gray "air" surrounding it with a label value of 1. This means that the transition in our method is much sharper than in the reference image.

3.2. Discussion

In this paper we have presented an approach to achieve smooth boundary transitions when rendering segmented data while significantly reducing the number of necessary texture lookups compared to methods based on full neighborhood analysis. The approach drastically reduces the amount of samples necessary to achieve the boundary smoothness,



(a) Nearest Neighbor (b) Two-Level Volume Rendering (c) Poor Man's Rendering (d) Intensites

Figure 4: Rendering a synthetic dataset with the Two-Level Volume Rendering (b) and the Poor Man's Segmentation (c) compared to the nearest neighbor filtering (a). (d) shows the values and the ordering of the intensity values for this dataset. As expected, the Two-Level Volume Rendering produces uniform boundaries regardless value difference. In the Poor Man's rendering, the boundary regions are dependent on the intensity value difference of the features. Both methods show a visual artefact in the area where three features coincide and the linear interpolation of green and blue results in yellow.

but does so at the cost introducing minor irregularities. The artifacts introduced by Poor Mans Rendering has two main visual manifestations. The first is an apparent widening of the transitional region between two segments. This behavior is predictable and can be minimized by intelligent selection of segment labels. For example, if a volume contains three segments but two of the segments never overlap, then the numerical label values can be selected such that the difference between two neighboring values never exceeds one, in which case any widening will be prevented. The second visual artifact is a non-smooth transition in neighborhoods heavily dominated by a single segment, such as the bottom of a depression. In such cases, the attenuation parameter is not guaranteed to reach zero before the nearest neighbor switches, resulting in a less smooth transition. It is possible to lower the impact of this artifact is to apply a stronger mapping in Equation 3.

In terms of performance, the value of the presented methods depends on a set of circumstances, some more predictable than others. For example, an application that is heavily bottlenecked by texture lookups is far more likely to see significant speedups than an application that spends most of its time of computations. On the other hand, the impact of the introduced irregularities will vary between different lighting and transfer function combinations which makes the assessment on the speed-vs-performance trade-off very much case dependent.

We believe our approach fills the gap between nearest neighbor class assignment and methods that rely on full neighborhood analysis. It could potentially be useful in cases where acceptable framerates have a high priority, such as previewing large datasets, interaction and transfer function design. The option to encode the segment membership together with the source data should also simplify rendering of segmented volume data on systems that may only have a single global transfer function.

3.3. Acknowledgments

David Karlsson, technical director at Interactive Institute, for pointing out the gap in the literature that this method fills. The presented concepts have been realized using the Voreen framework (www.voreen.org).

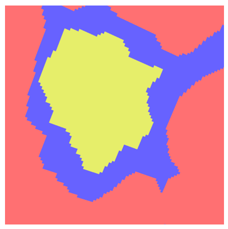
References

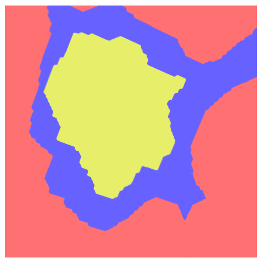
[HBH03] Hadwiger M., Berger C., Hauser H.: High-quality two-level volume rendering of segmented data sets on consumer graphics hardware. In *Proceedings of the conference on Visualization 2003* (2003), pp. 301–308. 1

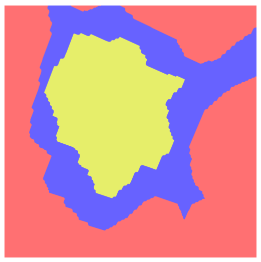
[HJ04] Hansen C. D., Johnson C. R.: *Visualization Handbook*, 1 ed. Academic Press, 2004. 1

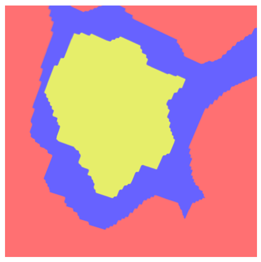
[KDC*00] Kaufman A., Dachille F., Chen B., Bitter I., Kreeger K., Zhang N., Tang Q., Hua H.: Real-time volume rendering. *International Journal of Imaging Systems and Technology 11*, 1 (2000), 44–52. 1

[UH00] Udupa J., Herman G.: *Three D Imaging in Medicine*. CRC PressINC, 2000. 1


(a) Nearest Neighbor


(b) Two-Level Volume Rendering


(c) Poor Man's Rendering


(d) Two-Level Volume Rendering without Label Volume

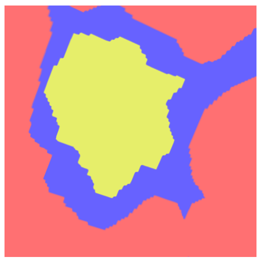
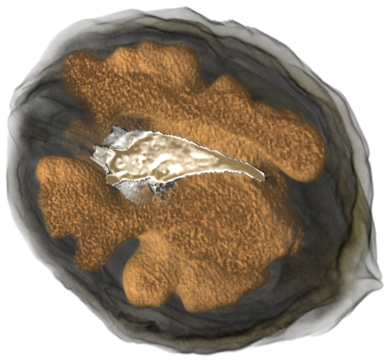
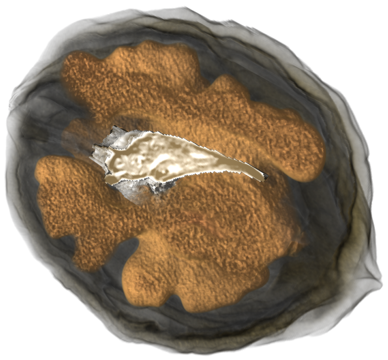
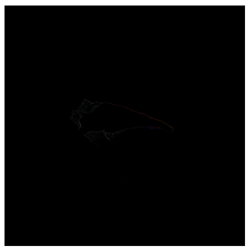

(e) Poor Man's Rendering without Label Volume

Figure 5: Comparative results of the presented techniques. (b) and (d) show the Poor Man's Rendering with and without a label volume respectively, while (a) and (c) are rendered using the Two-Level Volume Rendering approach. Note that while the both the Poor Man's Rendering and the Two-Level approach produce much smoother and visually pleasing results than the nearest neighbor filtering in (e), the differences between them is minimal.


(a) Two-Level Volume Rendering


(b) Poor Man's Rendering


(c) Difference Image

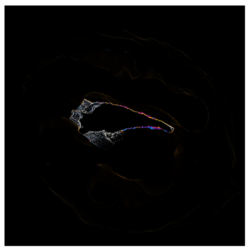

(d) Difference Image (x10)

Figure 6: Rendering the segmented Walnut dataset with our proposed method. (a) shows the reference image, created using the two-level volume rendering, while (b) is rendered using Poor Man's Rendering. (c) shows the absolute pixel-wise difference images between the two results and in (d) this difference is enhanced by one order of magnitude. As expected, the techniques differ only in the areas, where features touch whose label values differ by more than unity.