EUROGRAPHICS 2006 / E. Gröller and L. Szirmay-Kalos
(Guest Editors)

Volume 25 (2006), Number 3

Opacity Peeling for Direct Volume Rendering

Christof Rezk-Salama and Andreas Kolb

Computergraphik und Multimediasysteme, Universität Siegen, Germany

Abstract

The most important technique to visualize 3D scalar data, as they arise e.g. in medicine from tomographic measurement, is direct volume rendering. A transfer function maps the scalar values to optical properties which are used to solve the integral of light transport in participating media. Many medical data sets, especially MRI data, however, are difficult to visualize due to different tissue types being represented by the same scalar value. The main problem is that interesting structures will be occluded by less important structures because they share the same range of data values. Occlusion, however, is a view-dependent problem and cannot be solved easily by transfer function design. This paper proposes a new method to display different entities inside the volume data in a single rendering pass. The proposed opacity peeling technique reveals structures in the data set that cannot be visualized directly by one- or multi-dimensional transfer functions without explicit segmentation. We also demonstrate real-time implementations using texture mapping and multiple render targets.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

The majority of direct volume rendering techniques interpret the scalar field as a participating medium which emits and absorbs radiative energy at the same time. The optical properties required to solve the equations of light transport are obtained from the underlying scalar field by a user-specified transfer function.

The variety of existing volume rendering techniques may motivate the statement, that the volume rendering problem is solved. Indeed, this is true if we restrict ourselves to computing a solution of the volume rendering integral in real-time. The problem of deriving the optical properties required for ray integration, however, is still difficult to solve in practice. The process of transfer function design is often called *classification* because the assignment of optical properties to scalar data in general is a complex pattern recognition problem.

For volume data obtained by magnetic resonance imaging (MRI) the approach of extracting structures of interest by assigning optical properties based on the scalar value is often inadequate. One popular problem with volume rendering of MRI data is that different tissue types are represented by similar or overlapping ranges of scalar values. In the MRI data set shown in Figure 1 for example the brain cannot be visualized by pure transfer function assignment, because it is surrounded by skin and fat tissue whose intensities lie in the same range of data values. If the scalar values for the brain tissue are set to opaque, the surrounding soft tissue will become opaque as well and the brain will be occluded by these structures. Most approaches to solving this problem rely on explicit segmentation of structures of interest.

Classification in general is a time-consuming and cumbersome process. If adequate results are obtained they often cannot be shared between different data sets, not even between data sets of the same modality. However, as mentioned above the origin of the problems is occlusion, which is a view-dependent problem. In this paper we present an alternative way to resolve the above mentioned occlusion problems and circumvent the time-consuming process of classification. An example of the visualization results obtained by our method in real-time is displayed in Figure 2.

The presented visualization approach tries to solve occlusion problems without the use of segmentation techniques. Our solution targets time-critical scenarios, where volume rendering must be performed fast and almost autonomously, i.e with a minimum of user interaction.

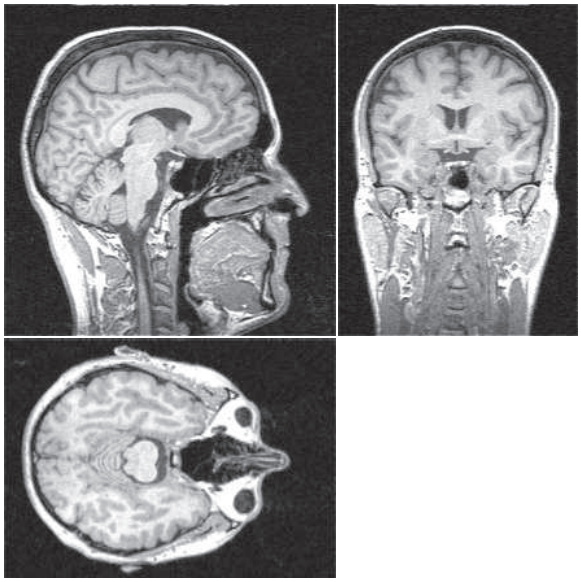

Figure 1: *Orthogonal slice images extracted from a typical MRI data set. Although different tissue types seem to be well separated in space, multiple different structures map to the same range of data values.*

Figure 2: *Example results of our opacity peeling technique applied to the MRI data set shown in Figure 1. The images show the first two layers obtained by our opacity peeling technique, rendered in real-time with on-the-fly gradient estimation and Blinn/Phong shading added. The second layer reveals internal structures, such as the brain, the eyes, muscle tissue, that are difficult to extract using pure transfer function design.*

The remainder of this paper is structured as follows. In Section 2 we have put together related work, relevant for this paper. In Secion 3, we briefly outline important theoretical aspects of direct volume rendering. Section 4 describes our new opacity peeling technique and Section 5 provides implementation details. The results of our technique are demonstrated in Section 6. Section 7 concludes and comments on future work.

2. Related Work

Many sophisticated techniques to solve the volume rendering integral in real-time have been proposed in the past, including the shear-warp algorithm [LL94], 3D texture slicing [CCF94, WGW94], 2D texture mapping [RSEB*00], pre-integration [EKE01], GPU ray-casting [KW03, RGWE03] or special purpose hardware [PHK*99]. Effective solutions have been introduced for rendering time-varying volume data [LMC01] and large data sets [Rod99, GWGS02] at interactive frame rates. A detailed overview of GPU-based volume rendering can be found in the SIGGRAPH course notes on Real-Time Volume Graphics [EHK*05].

At the current state-of-the-art, multi-dimensional transfer functions as proposed by Kniss et al. [KKH01] have proven superior to traditional 1D transfer functions. The magnitude of the first and second order derivatives of the scalar field are frequently used to expand the transfer function domain. Vega et al. show the benefit of 2D transfer functions for visualizing blood vessels in CT angiography data [VST*04]. Kniss et al. [KSW*04] use multi-dimensional transfer functions to classify coregistered multi-variate MRI data. Svakhine et al. [SES05] use 2D transfer functions for illustrative rendering. With multi-dimensional transfer functions, however, the assignment process becomes even more intricate.

Kniss et al. [KUS*05] propose a framework for rendering statistically classified volume data. Hadwiger et al. provide techniques for rendering segmented data [HBH03]. The Volume Shop application provided by Bruckner et al. [BG05] allows interactive segmentation of volumetric data with direct visual feedback. Other authors propose importance driven visualization [VKG04] and illustrative techniques [BGKG05], which may distinguish different tissue types by manual specification. In all cases high quality images are generated, but segmentation and classification is still time-consuming. Mora and Ebert use order-independent compositing to circumvent time-consuming classification [ME04].

The opacity peeling technique proposed in this paper is somehow related to depth peeling, a well-known technique in computer graphics for order-independent rendering of polygonal surfaces with non-refractive transparency [Eve01]. This technique has been adapted by Nagy and Klein to non-polygonal isosurface rendering in volume data [NK03]. They use depth-peeling to visualize the n closest isosurfaces with respect to the viewer in a single pass algorithm. Unlike this method, our technique uses the emission-absorption model for ray integration and is not build upon the assumption that interesting structures in the data can be adequately visualized by isosurface extraction.

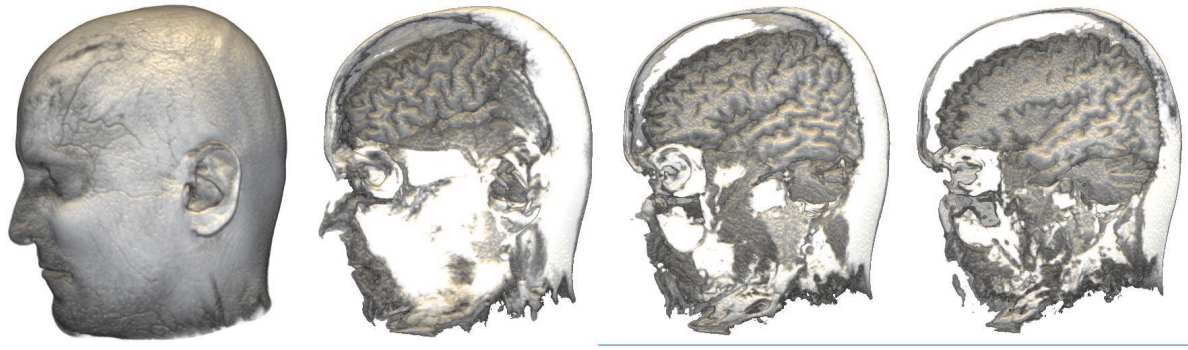

Figure 3: *Four simple opacity layers rendered for the MRI data set in Figure 1. The straight-forward idea of starting a new image, whenever the accumulated opacity value A_i exceeds a specified threshold leads to inadequate results. An additional threshold is required to obtain images such as the ones in Figure 2.*

3. Direct Volume Rendering in a Nutshell

Most approaches to direct volume rendering for scientific visualization are based upon a simplified model of light propagation in participating media. This model completely neglects scattering and only accounts for active emission and absorption. Light is assumed to travel along straight lines and this assumption allows us to integrate radiative energy along viewing rays.

The ray-casting algorithm calculates the radiance L that reaches the eye for every point on the image plane (i.e. for every pixel). In the following equations a ray of sight is parameterized by the scalar position s' along the straight line from the eye position through a point on the image plane. The ray is assumed to start at position s_0 sufficiently far away and reaches the eye at position s. The radiance $L(s)$ can then be calculated, according to

$$L(s) \;=\; L_0\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')\, e^{-\tau(s',s)} ds'. \qquad (1)$$

The first term on the right hand side is the background radiance L_0 at position s_0 which is absorbed along its way to the eye. The second term integrates additional light energy $q(s')$, which is emitted at a point s' along the ray and absorbed along its way from s' to the eye. The extinction τ along a ray segment is obtained by integrating the absorption coefficients κ, according to

$$\tau(s_0,s_1) \;=\; \int_{s_0}^{s_1} \kappa(t) dt. \qquad (2)$$

In practice, there are two methods to calculate numerical solutions of this rendering equations by iteration along the ray. Both approaches divide the viewing ray into small segments of fixed length and assume that emission and absorption are constant within each ray segment. The integrals in Equations 1 and 2 are substituted by Riemann sums and calculated by an iterative compositing scheme.

Back-to-front compositing starts with the background radiance L_0 at position s_0 and proceeds in direction to the eye,

$$L_i = q_i + (1-\alpha_i)\, L_{i-1}. \qquad (3)$$

Here, q_i and α_i refer to the source term and the opacity of the i-th ray segment.

The alternative compositing scheme is front-to-back compositing. It starts with zero radiance at the eye position and proceeds in direction away from the eye,

$$L_{i-1} \;=\; L_i + (1-A_i)\, q_i, \qquad (4)$$

$$A_{i-1} \;=\; A_i + (1-A_i)\, \alpha_i. \qquad (5)$$

The value A_i is the opacity gathered along the ray so far and must be updated in lock-step with the radiance. The advantage of back-to-front compositing is that it does not require this additional value A_i. Front-to-back compositing, however, is advantageous for ray-casting because the calculation of a viewing ray can be stopped if the opacity A_i approaches one. The radiance value will not change any further if A_i is close to one. The reason for this is that the contributions of all further ray segments are actually occluded. Such a technique is known as *early ray termination*.

4. Opacity Peeling

As described in Section 1, a major problem with rendering of MRI data is that interesting structures might be occluded by surrounding structures of less importance. In the previous section we have already seen a measure, that allows us to determine when occlusion occurs along a ray. If we use front-to-back compositing and the accumulated opacity A_i in Equation 5 approaches one, all further ray segments will be occluded.

The initial idea behind our opacity peeling technique is quite simple. We use a direct volume rendering approach

with front-to-back compositing. Whenever the accumulated opacity value A_i exceeds a specified threshold, say 0.99, early ray termination would stop the calculation for the viewing ray. Instead of terminating the ray, however, we reset the accumulated opacity to zero and start a new image at this position. We illustrate this technique by the pseudo code displayed in Listing 1. The values `L[j]` represent the radiance contribution to the opacity layer `j`. We use a single floating point value in this example for simplicity. For color images `L` and `q(s)` should be a RGB triplets. A new layer is started if the accumulated opacity `A` exceeds the user-specified threshold `T_high`.

Such a simple technique generates layered images as displayed in Figure 3. Four opacity layers are shown which contain the contribution of different ray segments. The second image shows structures occluded by the first image, the third image is occluded by the second one and so on.

If we look at the resulting images, we immediately notice a problem with this initial approach. The original intension was to peel off the skin and fat layer in order to visualize the brain tissue behind it. In Figure 3 the skin and fat layer is peeled off only partially. None of the layers shows a clear image of the brain surface. In the second layer parts of the fat tissue still occludes the brain, while in the third and fourth layers portions of the brain have already been peeled off. The reason for this is the varying thickness of the fat tissue. Obviously the rays reach the opacity threshold at different positions and this cannot be fixed by modifying the opacity threshold.

There is another problem with this approach. We start a new image when the accumulated opacity is high. If this occurs in a region within the volume data where the opacity α of the current ray segment itself is close to one, we will end up starting new layers with every integration step until the ray exits the region of high opacity.

```
int   j    = 0; // layer index
float A    = 0; // accumulated opacity

L[0] = 0; // radiance 1st layer

// for all ray segments
for(int s = 0; s < s_max; ++s) {

    // front-to-back compositing
    L[j] = L[j] + (1-A) * q(s);
    A    = A    + (1-A) * alpha(s);

    if (A > T_high) {
        j++;      // start a new layer;
        A    = 0; // reset opacity
        L[j] = 0; // radiance new layer
    } // if
} // for
```

Listing 1: *Pseudo code for ray tracing with opacity layers.*

Fortunately, there is a simple solution to both problems. We introduce another threshold `T_low`. The opacity value α of the current ray segment must fall below this threshold, otherwise we do not start a new layer. The code sample in Listing 1 can easily be modified to account for this additional threshold (Listing 2). In any case the additional threshold solves the problem of continuous layer restarts in opaque regions.

```
if ((A > T_high) && (alpha(s) < T_low))
{
    // start a new layer;
}
```

Listing 2: *Modification to Listing 1 introducing a second threshold `T_low`.*

If we can assume that different tissue types are separated by a thin region of low opacity, the second threshold also solves the problem of varying tissue thickness. For most MRI sequences, which measure low signal for air and fluid, this is true (assuming that opacity is assigned proportional to the data value). However, if this is not the case, the gradient vector of the scalar field may be estimated for each voxel as will be outlined in Section 4.1. The gradient magnitude can be used instead of the opacity α. A new layer is then started when the gradient magnitude is larger than a specified threshold. To improve the robustness of the gradient magnitude for boundary detection, it can be divided by the magnitude of the second order derivative (Marr-Hildreth edge detection [MH80]) as proposed by Kindlmann et al [KD98]. This boundary detection technique should be used for MRI sequences that represent fluid by high intensity, such as MR-CISS or MR-FISP sequences.

The data set in Figure 1 is an example for an MRI sequence where a lower threshold on opacity α works well. Figure 4 show image results for the second layer with different thresholds `T_low`. In the leftmost image, the threshold is too low, which results in holes caused by rays that never reach the second layer. The rightmost image demonstrates that the lower threshold is sensitive enough to even separate the skin from the muscle tissue. All images were generated with an upper threshold of `T_high`$= 0.95$.

4.1. Multisampling

One of the major requirements for our visualization technique was to perform well for MRI data obtained within the restrictions of a real clinical workflow, not only for ideal data measured on healthy volunteers. Unlike many ideal MRI data sets found in the internet, real patient data obtained in clinical environments often have strict limits with respect to

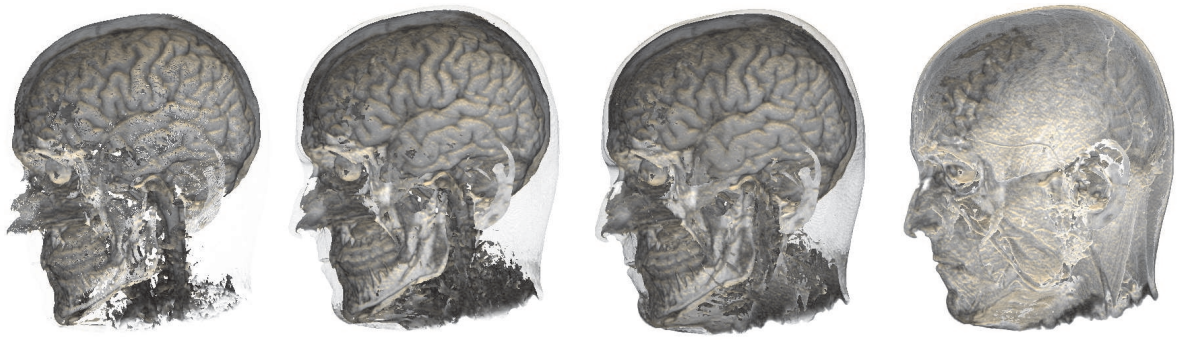

Figure 4: *The second layer images for different lower thresholds applied to the MRI data from Figure 1. The lower threshold increases from left to right. In the leftmost image the threshold is too low resulting in visible holes. The rightmost image shows that the opacity peeling technique even allows removing the skin layer to reveal the muscle tissue behind it, which is extremely hard to obtain by explicit segmentation.*

the acquisition time and the period of time an elderly patient can lie still during the measurement. Such data is often noisy and may contain visual artifacts due to patient motion. Noise is a problem especially for intraoperative data measured by open MR devices during a surgical intervention.

The opacity peeling technique described above is quite sensitive to noise, due to the sharp thresholds applied, which often causes ray segments for adjacent pixels to terminate at considerably different positions. To alleviate this problem for noisy data, we use a multi-sampling technique to average the opacity values of adjacent rays. The alpha values for each position along the ray are then determined by averaging multiple samples slightly shifted around the ray position at locations on the plane perpendicular to the viewing ray. The sampling patterns for $5\times$ and $9\times$ multisampling are illustrated in Figure 5 and the results are shown in Figure 6. The distance between the individual sampling points is equal to the size of a voxel.

If one of the multi-sampling schemes is applied anyway, we can take two additional samples along the ray direction and use them to compute central differences for on-the-fly gradient estimation. This allows us to integrate local illumination techniques such as the Blinn/Phong model on a per-fragment basis.

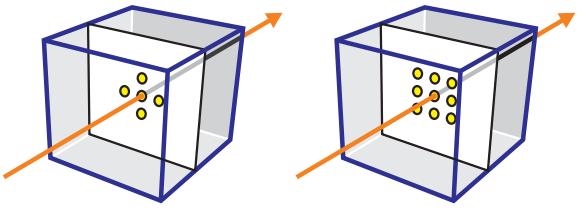

Figure 5: *Multisampling schemes. Multiple samples of the volume data are obtained at locations lying on a plane perpendicular to the ray. Left:* $5\times$ *multisampling. Right:* $8\times$ *multisampling.*

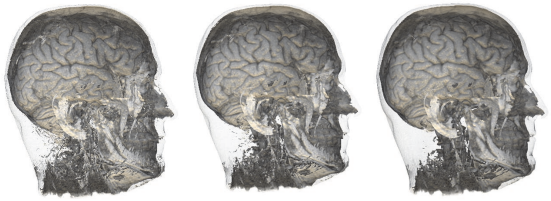

Figure 6: *Rendering results for the second layer obtained by multisampling. Left: no multisampling. Middle:* $5\times$ *multisampling. Right:* $9\times$ *multisampling. The multisampling techniques significantly reduces noise in the data*

If multisampling the data on-the-fly is too slow, the data set can alternatively be filtered with a tent filter or a gaussian as a preprocessing step. The gradient vectors for local illumination can as well be pre-computed. However, we prefer on-the-fly multisampling for two reasons: The first reason is that prefiltering is independent of the viewing direction, while our multi-sampling scheme only smoothes the data perpendicular to the viewing ray. Second, we can use the original sample values for luminance and use the smoothed value only for alpha and for thresholding. Figure 7 shows the the first and second layer of an preoperative MRI data set measured immediately before a surgical intervention, rendered with $9\times$ multisampling and Blinn/Phong illumination.

4.2. Transfer Function Design

Another requirement for our technique was to work almost autonomously, i.e. without a high amount of user interaction to specify optical properties. As a first simplification, we restrict the emission values q to a single luminance value in-

stead of RGB triplets. This simplification is not a limitation imposed by our technique, but is motivated by the preferences of many radiologists. They prefer monochrome display of the data, because this is the way they are used to when looking at slice images. The subtle shades of color seen in some of the example images are caused by shading with colored light sources and not by classification.

The luminance and alpha values required for ray-integration are specified as a linear mapping of the scalar value,

$$q(s) = \frac{s - s_{\min}}{s_{\max} - s_{\min}}; \quad (6)$$

$$\alpha(s) = \frac{s - t_{\min}}{t_{\max} - t_{\min}}; \quad (7)$$

Instead of a complex transfer function, we found that the four thresholds $s_{\min}$, $s_{\max}$, $t_{\min}$ and $t_{\max}$ are sufficient to render typical MRI data if occlusion problems are resolved using opacity peeling. An important benefit of this simple transfer function setup is that most clinicians are already used to it, because it is very similar to the grayvalue windowing for slice images. The graphical user interface designed for generating all the example images in this paper consists of nothing more than six sliders, two for luminance and alpha, respectively, and two for the upper and the lower threshold for opacity peeling.

5. GPU Implementation

The described opacity peeling technique can easily be implemented in any existing framework for software ray-casting. Only very few modifications to the code are necessary, such as the ones outlined in Listing 1 and 2. For real-time rendering the presented technique can be used to supplement existing GPU implementations using texture-slicing or GPU raycasting.

Figure 7: *The proposed opacity peeling technique with multisampling, is capable of revealing important structures from preoperative MRI data, such as the blood vessels and brain tissue. Note that the first layer clearly shows the fiducial markers used during the intervention which can be used for co-registration with real video data. The second level reveals veins and brain structures, which is important information for the surgeon*

5.1. Texture Slicing

The most popular texture-based volume rendering techniques are 3D texture slicing (usually with viewport-aligned slices, [WGW94]) or the 2D multi-texture based approach (with object-aligned slices, [RSEB*00]).

Opacity peeling can be used with both techniques. Our GPU-based implementation requires hardware support for multiple render targets (MRT), which allow a fragment program to output multiple color values in one pass. The volume is rendered slice by slice in front to back order. For compositing we use a double-buffered offscreen rendering technique known as ping-pong blending [KPHE02]. In one step we read from the first buffer and write to the second one. In the next step the buffers are swapped before the subsequent slice is drawn. This allows the alpha blending step required for compositing to be performed in the fragment shader at floating point precision. It replaces standard alpha blending, performed at 8bit fixed-point precision in the frame buffer operations.

In order to compute four depth layers in one rendering pass, we simultaneously bind four floating point rendering targets using the frame buffer objects OpenGL extension. On many graphics boards, it is also possible to use the same buffer both for reading and writing, as mentioned by Engel [EHK*05]. We must take care ourselves to avoid concurrent read-write access between different fragment processing units. In our case this is no problem because we are reading and writing to the same pixel location. We can simply replace the double MRT buffers by a single MRT buffer and save a considerable amount of local video memory.

A Cg fragment program implementing opacity peeling with four layers in one step is shown in 3. Multi-sampling and shading was omitted in this example to keep the code simple. The struct at the top of the listing defines the output value of the fragment program, which consists of four RGBA quadruplets, one for each of the render targets. The same buffers are simultaneously bound as 2D texture images `Buffer0`–`Buffer3` for read access in the fragment program.

The four thresholds $s_{\min}$, $s_{\max}$, $t_{\min}$ and $t_{\max}$ are stored in one RGBA quadruplet called `TF`. After sampling the volume and the render targets, the luminance and opacity thresholds are applied. The alpha values `A0` – `A3` are the alpha blending weights for the four targets, which are initialized with zero. The following if-clauses check to which buffer we are currently rendering and set the alpha values accordingly. At the end of the code the compositing is actually performed. Note that we are using associated colors, which means that RGB components must be pre-multiplied with opacitiy A. The displayed code can be modified to account for multi-sampling, on-the-fly gradient estimation and shading.

5.2. GPU Ray-Casting

A different strategy to perform volume rendering on the GPU is to exploit the support for dynamic loops and branches to implement ray-casting [RGWE03, KW03]. The first step in such an implementation is to calculate the starting points for the rays by rasterizing the front faces of the bounding box or an octree hierarchy of bounding boxes [HSS*05]. Afterwards the z-buffer is used as a depth texture to obtain the correct ray-starting points. In such an implementation we can integrate opacity peeling without multiple render targets. The idea is to perform GPU-raycasting as described in the original implementations. Whenever our restart condition decides to start a new layer, we simply update the z-buffer and exit the fragment program. The next layer is calculated by using the updated depth-texture to start the new layer at exactly the position where the previous layer has terminated. Note that although the GPU ray-casting procedure must be restarted once for each layer, the volume is only traversed once from front to back.

```
struct COLOROUT {
    float4 dst0 : COLOR0;
    float4 dst1 : COLOR1;
    float4 dst2 : COLOR2;
    float4 dst3 : COLOR3;
};

COLOROUT main(half3     uvw     : TEXCOORD0,
              half3     sceenpos: TEXCOORD1,
      uniform sampler3D DataSet : TEXTURE0,
      uniform sampler2D Buffer0 : TEXTURE1,
      uniform sampler2D Buffer1 : TEXTURE2,
      uniform sampler2D Buffer2 : TEXTURE3,
      uniform sampler2D Buffer3 : TEXTURE4,
      uniform float     T_low,
      uniform float     T_High,
      uniform float4    TF)
{
    COLOROUT retval;
    // sample 3D texture
    float s = tex3D(DataSet,uvw);
    // sample MRT targets
    float4 dst0 = tex2D(Buffer0,screenpos.xy);
    float4 dst1 = tex2D(Buffer1,screenpos.xy);
    float4 dst2 = tex2D(Buffer2,screenpos.xy);
    float4 dst3 = tex2D(Buffer3,screenpos.xy);

    float4 src;
    src.rgb = clamp((s-TF.g)/(TF.r-TF.g),0.0,1.0);
    src.a = 1.0;
    float A = clamp((s-TF.a)/(TF.b-TF.a),0.0,1.0);

    float A0,A1,A2,A3;
    A0 = A1 = A2 = A3 = 0.0;

    if (dst1.a <= 0.0) { // blend to layer 0
          A0 = A;
    }
    if (dst2.a <= 0.0) { // blend to layer 1
        if (((dst0.a > T_high) && (A < T_low)) ||
            (dst1.a > 0.0)) {
          A1 = A;
        }
    }
    if (dst3.a <= 0.0) { // blend to layer 2
        if (((dst1.a > T_high) && (A < T_low)) ||
            (dst2.a > 0.0)) {
          A2 = A;
        }
    }
    if (((dst2.a > T_high) && (A < T_low)) ||
         (dst3.a > 0.0)) {
          A3 = A;       // blend to layer 3
    }

    retval.dst0 = dst0 + (1.0 - dst0.a) * src * A0;
    retval.dst1 = dst1 + (1.0 - dst1.a) * src * A1;
    retval.dst2 = dst2 + (1.0 - dst2.a) * src * A2;
    retval.dst3 = dst3 + (1.0 - dst3.a) * src * A3;

    return retval;
}
```

Listing 3: *Cg fragment shader for texture slicing with opacity layers using multiple render targets.*

6. Results

The described algorithms were implemented on an ATI Radeon X850 graphics board with 256 MB memory. The performance achieved lie between 5 and 20 frames per second, depending on viewport size, sampling distance, and the complexity of the fragment program (multi-sampling, shading).

We do not claim that opacity peeling is applicable to any kind of volume rendering problem or data type. Neither do we claim that this technique can replace segmentation or classification techniques, if high accuracy is required. The proposed approach has been developed with the time-critical visualization scenarios in mind, where structures inherent in the data must be visualized fast and effectively with only a minimum of user interaction.

One of those scenarios is neurosurgery where intraoperatively acquired MRI data is valid only a very limited period of time after acquisition and must thus be visualized immediately, without much time for preprocessing or segmentation. We have tested our rendering technique with a considerable number of pre- and intraoperative patient data sets from clinical practice. We claim that the developed technique fulfills the requirement of fast and almost autonomous visualization. All the result images in this paper have been obtained with only a few seconds spend for threshold adaptation. The opacity peeling technique performs best if the structures in the data are onion-like. Generated images are not free of visual artifacts, as can be seen at the ear in Figure 4.

Figure 9 shows four opacity layers rendered for a noisy intraoperative data set. The first layer shows the skin of the patient. Although this first layer does not reveal any relevant anatomical information, it can be used to co-register the fiducial markers attached to the skin to intraoperative video recordings. The second layer clearly reveals the eyeballs, the meninx, vessels and surrounding muscle tissue. The third layer peels off the meninx and reveals the brain. The fourth

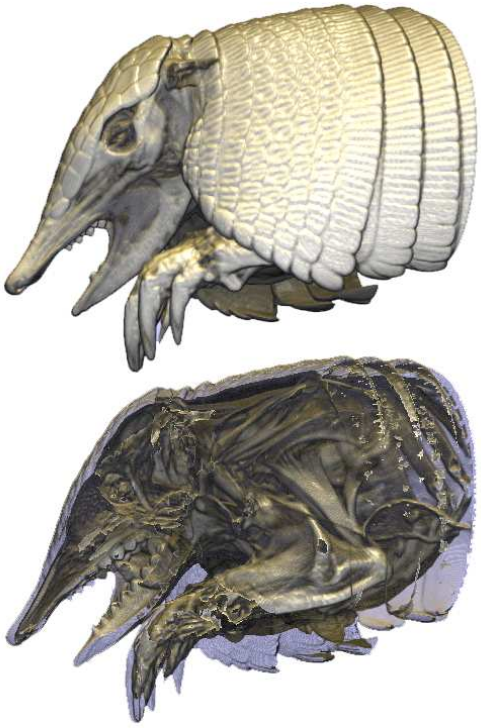

Figure 8: *Direct volume rendering of the Armadillo CT data set from the UTCT data archive. First two opacity layers. The second layer shows bone structures and muscle tendons hidden behind the opaque shell. The original data set available at* `http://utct.tacc.utexas.edu/` *was downsampled to* $512^2 \times 499$*, 12 bit*

.

layer shows deeper structures of the brain. The high threshold had to be decreased considerably to create these images. In order to visualize the deeper structures the upper threshold must be decreased. A similar data set is shown in Figure 10. This is an intraoperative data set from a tumor resection recorded immediately before opening the head. The third and fourth layer clearly reveal the pathological structures and allow the evaluation of the extent of the tumor.

Additionally, the opacity peeling technique was applied to CT data. Figure 8 shows the first two layers for the Armadillo data set available at the UTCT data archive (`http://utct.tacc.utexas.edu/`). In the second layer the shell of the animal was peeled off revealing bones and muscle tendons. The same data set with different thresholds is displayed in Figure 11.

Throughout our experiments with different data sets, especially MRI data, we have found that the simple opacity peeling technique allows us to reveal complex structures hidden in the data. We were often surprised when discovering structures we did not see before inside data sets that we were working with for quite some time.

7. Conclusion and Future Work

We have explained and demonstrated the novel technique of opacity peeling, which can be applied to reveal interesting structures hidden in volumetric data, that cannot be easily visualized by transfer function assignment. We have shown that the proposed technique fulfils the requirements of fast and autonomous visualization.

The techniques introduced in this paper are meant as building blocks for engineering customized rendering procedures for application-specific visualization tasks. We strongly believe that this is the future direction in scientific visualization. Effective techniques must be tailored to the concrete visualization problem by merging the expertise of the computer scientist with the data-specific knowledge of the client to build solutions that are intuitive to use for both of them.

The opacity peeling technique was designed in the first place to provide a volume rendering technique that performs fast and almost autonomously in clinical practice. Our strategy for transfer function assignment was driven by this goal. However, other strategies are thinkable. In some cases it might be useful to apply more complex transfer functions, and possibly change the transfer function for each layer. This would result in applications where classification accounts for ray history. After volume rendering different layers can be composited in image spaces, probably with image-based enhancements known from non-photorealistic rendering.

8. Acknowledgements

The clinical data sets were generously provided by Peter Hastreiter (Neurocenter) and Christopher Nimsky (Department of Neurosurgery) at the University of Erlangen-Nuremberg.

References

[BG05] BRUCKNER S., GRÖLLER E.: VolumeShop: An Interactive System for Direct Volume Illustration. In *Proceedings of IEEE Visualization* (2005).

[BGKG05] BRUCKNER S., GRIMM S., KANITSAR A., GRÖLLER E.: Illustrative Context-Preserving Volume Rendering. In *Proceedings of EuroVis 2005* (2005), pp. 69–76.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated Volume Rendering and Tomographic Reconstruction using Texture Mapping Hardware. In *Proceedings of IEEE Symposium on Volume Visualization* (1994), pp. 91–98.

[EHK*05] ENGEL K., HADWIGER M., KNISS J.,

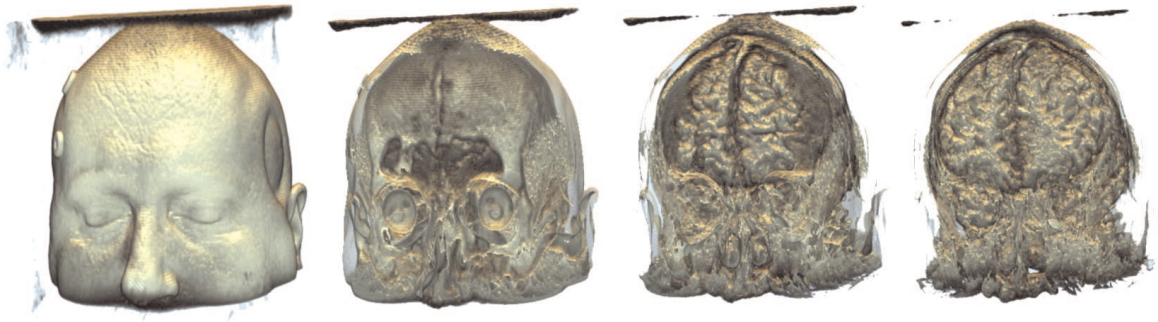

Figure 9: *Image results obtained by rendering intraoperative MRI data (*$256^2 \times 112$*, 12 bit). Four opacity layers created in a single rendering pass. The second layer shows the eyes, the meninx and muscle tissue. The third layer peels off the meninx and reveals the brain surface. The fourth layer shows deeper structures of the brain. The high threshold had to be decreased considerably to create these images.*

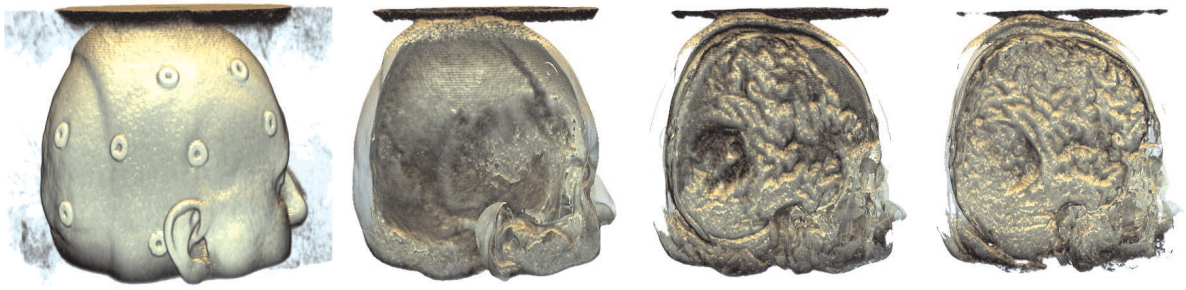

Figure 10: *Image results obtained by rendering intraoperative MRI data (*$256^2 \times 112$*, 12 bit). Four opacity layers created in a single rendering pass. The first layer displays the skin of the patient and does not show valuable anatomical information, but can be used for coregistration based on the fiducial markers. The second layer shows the meninx and muscle tissue. The tumor is clearly visible in the 3rd layer. The 4th layer displays the boundary of the tumor and deeper structures of the brain.*

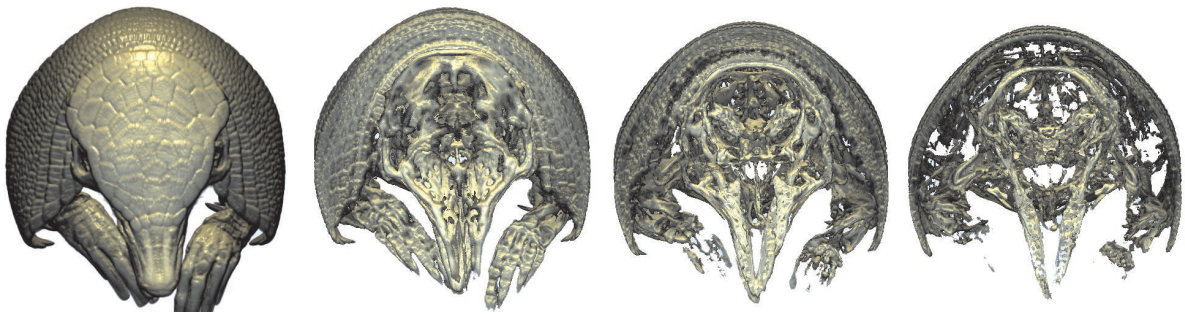

Figure 11: *Opacity peeling applied to a CT data set.Direct volume rendering of the Armadillo data set from the UTCT data archive (*`http://utct.tacc.utexas.edu/`*). Opacity peeling is used to remove the shell and different layers of bone structures. The original data set was downsampled to* $512 \times 512 \times 499$*, 12 bit*

LEFOHN A., REZK-SALAMA C., WEISKOPF D.: Real-Time Volume Graphics. In *ACM SIGGRAPH Course Notes* (2005).

[EKE01] ENGEL K., KRAUS M., ERTL T.: High-Quality Pre-Integrated Volume Rendering Using Hardware-Accelerated Pixel Shading. In *Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2001).

[Eve01] EVERITT C.: *Interactive Order-Independent Transparency*. White paper, NVIDIA, 2001.

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive Rendering of Large Volume Data Sets. In *Proceedings of IEEE Visualization* (2002), pp. 53–60.

[HBH03] HADWIGER M., BERGER C., HAUSER H.: High-Quality Two-Level Volume Rendering of Segmented Data Sets on Consumer Graphics Hardware. In *Proceedings of IEEE Visualization 2003* (2003), pp. 301–308.

[HSS*05] HADWIGER M., SIGG C., SCHARSACH H., BÜHLER K., GROSS M.: Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. In *Proceedings of Eurographics* (2005), pp. 303–312.

[KD98] KINDLMANN G., DURKIN J.: Semi-automatic Generation of Transfer Functions for Direct Volume Rendering. In *Proceedings of IEEE Symposium on Volume Visualization* (1998), pp. 79–86.

[KKH01] KNISS J., KINDLMANN G., HANSEN C.: Interactive Volume Rendering using Multi-dimensional Transfer Functions and Direct Manipulation Widgets. In *Proceedings of IEEE Visualization* (2001), pp. 255–262.

[KPHE02] KNISS J., PREMOZE S., HANSEN C., EBERT D.: Interative Translucent Volume Rendering and Procedural Modeling. In *Proceedings of IEEE Visualization* (2002).

[KSW*04] KNISS J., SCHULTZE J., WÖSSNER U., WINKLER P., LANG U., HANSEN C.: Medical applications of multi-field volume rendering and vr techniques. In *Proc. Eurographics/IEEE TCVG Symposium on Data Visualization* (2004).

[KUS*05] KNISS J., UITERT R. V., STEPHENS A., LI G., TASDIZEN T., HANSEN C.: Statistically Quantitative Volume Visualization. In *Proceedings of IEEE Visualization* (2005).

[KW03] KRÜGER J., WESTERMANN R.: Acceleration Techniques for GPU-based Volume Rendering. In *Proceedings of IEEE Visualization 2003* (2003), pp. 287–292.

[LL94] LACROUTE P., LEVOY M.: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *Proceedings of ACM SIGGRAPH* (1994), pp. 451–458.

[LMC01] LUM E. B., MA K. L., CLYNE J.: Texture Hardware Assisted Rendering of Time-Varying Volume Data. In *Proceedings of IEEE Visualization* (2001), pp. 263–270.

[ME04] MORA B., EBERT D.: Instant volumetric understanding with order-independent volume rendering. *Computer Graphics Forum 23*, 3 (2004).

[MH80] MARR D., HILDRETH E.: Theroy of edge detection. In *Proc. Royal Society of London,B 207:187-217* (1980).

[NK03] NAGY Z., KLEIN R.: Depth-Peeling for Texture-Based Volume Rendering. In *Proceedings of Pacific Graphics* (2003), pp. 429–433.

[PHK*99] PFISTER H., HARDENBERGH J., KNITTEL J., LAUER H., SEILER L.: The VolumePro real-time ray-casting system. In *Proceedings of ACM SIGGRAPH* (1999), pp. 251–260.

[RGWE03] RÖTTGER S., GUTHE S., WEISKOPF D., ERTL T.: Smart Hardware-Accelerated Volume Rendering. In *Procceedings of EG/IEEE TCVG Symposium on Visualization VisSym '03* (2003), pp. 231–238.

[Rod99] RODLER F. F.: Wavelet Based 3D Compression with Fast Random Access for Very Large Volume Data. In *Proceedings of Pacific Graphics* (1999), p. 108.

[RSEB*00] REZK-SALAMA C., ENGEL K., BAUER M., GREINER G., ERTL T.: Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization. In *Proceedings of ACM SIGGRAPH/Eurographics Workshop on Graphics Hardware* (2000).

[SES05] SVAKHINE N., EBERT D., STREDNEY D.: Illustration motifs for effective medical volume illustration. *IEEE Computer Graphics and Applications 25*, 3 (May 2005), 31–39.

[VKG04] VIOLA I., KANITSAR A., GRÖLLER E.: Importance-Driven Volume Rendering. In *Proceedings of IEEE Visualization* (2004), pp. 139–145.

[VST*04] VEGA HIGUERA F., SAUBER N., TOMANDL B., NIMSKY C., GÜNTHER G., HASTREITER P.: Automatic Adjustment of Bidimensional Transfer Functions for Direct Volume Visualization of Intracranial Aneurysms. In *Proceedings of SPIE Medical Imaging* (2004).

[WGW94] WILSON O., GELDER A. V., WILHELMS J.: *Direct Volume Rendering via 3D-textures*. Tech. Rep. UCSC-CRL-94-19, Univ. of California, Santa Cruz, 1994.