Depth-Peeling for Texture-Based Volume Rendering

Zoltán Nagy, Reinhard Klein
Institut für Informatik II, Bonn University
Römerstraße 164, D-53117 Bonn, Germany
{zoltan|rk}@cs.uni-bonn.de

Abstract

We present the concept of volumetric depth-peeling. The proposed method is conceived to render interior and exterior iso-surfaces for a fixed iso-value and to blend them without the need to render the volume multiple times. The main advantage of our method over pre-integrated volume rendering is the ability to extract arbitrarily many iso-layers for the given iso-value. Up to now, pre-integrated volume rendering is only capable of visualizing the nearest two (front and back-faced) iso-surfaces. A further gain of our algorithm is the rendering speed, since it does not depend on the number of layers to be extracted, as for previous depth-peeling methods. We rather exploit the natural slicing order of 3D texturing to circumvent the handicap of storing intermediate layers in textures, as done in polygonal-based depth-peeling approaches. We are further capable of rapidly previewing the volume data, when only few context information about the concerning dataset is available. An important example of use in the area of non-photorealistic rendering is given, where we can distinguish between visible and hidden silhouettes, which are important elements in stylization. By using standard OpenGL extensions, we allow the exploration of spatial relationships in the volume -at interactive rates- in hardware.

1. Introduction

This work deals with the question of how to extract and visualize the n-th closest iso-surface to the viewer, based on the *same iso-value*, for a given volumetric dataset. To resolve the problem, we use the idea of *depth-peeling* and adopt it to volumetric representations. By exploiting the anatomy of iso-surfaces in volumes, the proposed scheme clearly beats methods like *depth-sorting*, *screen-door transparency* and the primary *depth-peeling* approach, since the result is rendered in a single pass without further context information. By using transfer functions, this cannot be accomplished, since transfer functions influence either all or none of the samples of the inspected volume. Consider, for instance, some hollow spheres with a shell thickness d>0, recursively nested into each other, with a shell opacity of α_{shell}. A transfer function would either mask out all shells or none of them, since ordinary transfer functions hardly offer the possibility of capturing these local spatial properties of the matter. Higher-order transfer functions, e.g. like in [13], do not help here, since the layer count information is not provided for a particular view. We solve the problem by appropriately cumulating layer information in the render target and combining the result in an adequate manner.

2. Related Work

Since pioneer work [12], transparency has been used in volume visualization for better conveying shape [8, 9, 10], implicitly in generating point-stippling techniques [14, 15, 16] as well as in NPR for illustrating superimposed, differently shaded iso-surfaces [17]. The *bubble model* [2] uses a similar previewing scheme like our model, but we are able to distinguish structures for a fixed iso-value. Two-Level Volume Rendering(2lVR) as described in [6] -similar to our method- is designated to visualize the volumetric data in a *Focus+Context(F+C)* manner; this method, however, requires pre-classification by segmentation or transfer functions and is not hardware-accelerated in that implementation.

Depth-peeling was realized in [18] using *Virtual Pixel Maps* and *Dual Depth Buffers* [3]. Later on, special hardware functionality accelerated the use of depth peeling [5]; also its use in NPR and volume clipping was soon recognized [4, 21]. The common idea of this methods is depth-testing with two depth buffers and two belonging depth tests. The first depth test can be imagined (depending on the implementation) as the standard GL_LEQUAL testing. The second depth test, in turn, further rejects the active fragment, if it is nearer to the viewer than the depth of the pixel rendered at the same screen position in the previous pass. Our way of extracting the different layers is slightly varied.

3. Volumetric Depth-Peeling

Consider a volume and a ray casted through it. We call the number n of penetrated iso-surfaces, alternating between interior and exterior boundaries, the *n-th layer*. We regard a fragment on an iso-surface as *exterior (interior)*, if the corresponding gradient is front- (back-) facing. This differs slightly from the original definition of depth-peeling, where polygonal representations of objects were used. We rather treat objects to have iso-boundaries with a thickness larger than zero -which is typical for volume representations.

The idea for counting (iso-surface-)layers during the rendering process is closely related to the method of *shadow volumes* [1, 7]. The main differences of iso-surface counting in our sense differ from shadow volumes by

- changing the value in the stencil buffer, if the iso-surface is penetrated -instead of the polygon,
- remembering, which iso-surface was penetrated last,
- blending of intermediate lit surfaces, if desired.

To achieve the aims of this paper, the above steps have to mapped appropriately to the build-in capabilities of the video card. Unfortunately, a simple solution by using an appropriately configured alpha or stencil test, respectively, is excluded for two reasons. First, hardware assisted alpha tests are configurable only on per-primitive, rather than on per-fragment basis. Second, it is currently not possible to address the stencil counter programmatically for registering the active iso-surface number from fragment programs.

We are thus forced to impose a workaround by simulating the alpha-test by a fragment program and storing intermediate rendering results in a window-aligned 2D-texture. Using the observations from the previous subsection, we can simply rewrite the shadow volume approach for the extraction of the n-th layer. During the rendering session, the texture units are organized as in table 1.

Texture Unit	Format	Content
0: Dataset (3D)	RGBA	24 Bit $\overrightarrow{g_f}$, 8 Bit α
1: Framebuffer copy(2D)	RGBA	24 Bit c_{old}, 8 Bit layer#
2: Layer colors (1D)	RGB	24 Bit c_{layer}
3: View direction	—	$\overrightarrow{v_f}$

Table 1. *Texture setup*.

The main steps for extracting the n-th layer can be depicted as in figure 1. The algorithm works by rendering the slices of the 3D texture in texture unit (TU) 0 and copying

For each fragment of the current slice:

1. **Determine α-test.**
   sign $\leftarrow (-1)^{layer\#+1}$
2. **α-test.**
   $sign \cdot \alpha < sign \cdot \alpha_{iso}$?
   Yes: Kill fragment.
   No: layer $\leftarrow$ layer# + 1;
3. **Lighting.**
   $$c_{new} \leftarrow c_{spec}\Big(\overrightarrow{g_f} \cdot \frac{\overrightarrow{l}+\overrightarrow{v_f}}{\|\overrightarrow{l}+\overrightarrow{v_f}\|}\Big)^{ex} + c_{diff}(\overrightarrow{g_f} \cdot \overrightarrow{l}) + c_{amb}$$
4. **Coloring.**
   $c_{new} \leftarrow c_{new} \otimes c_{layer}$
5. **Blending.**
   $$c_{new} \leftarrow \begin{cases} c_{old} & : \quad layer < 2 \\ (1-\frac{1}{layer})c_{old} + \frac{1}{layer}c_{new} & : \quad layer \geq 2 \end{cases}$$
6. **Layer test.**
   $$c_{result} \leftarrow \begin{cases} \boxed{c_{back} \mid \text{layer}} & : \quad layer < n \\ \boxed{c_{old} \mid \text{layer}} & : \quad layer > n \\ \boxed{c_{new} \mid \text{layer}} & : \quad layer = n \end{cases}$$

Figure 1. *Survey of the fragment program for ordinary depth-peeling.* $\otimes$ denotes component-wise multiplication, α_{iso} the iso-value, c_{spec}, c_{diff} and c_{amb} specular, diffuse and ambient weights, ex the specular exponent, c_{back}, c_{old} and c_{new} the background color, the color in the framebuffer and the resulting color, respectively.

the content of the framebuffer in TU 1 alternatingly. The latter step answers the purpose of accessibility, since fragment programs are currently not able to read the framebuffer directly. Initially, TU 1 is cleared with the background color c_{back}, annotating in the α-channel that no layer was extracted so far. The interior loop (steps 1-6) describes the action taking place on per-fragment basis.

In step 1, -depending if the last layer was odd or even- we classify, wether currently an interior or exterior boundary has to be extracted. Step 2 immediately rejects fragments from further processing, if the fragment does not lie on an iso-surface boundary. The surviving fragments are supposed to be located intuitively behind the increased layer. Step 3 prophylactically calculates the canonical Phong-lighting for a local viewer. The view direction $\overrightarrow{v_f}$ is obtained on a per-fragment basis by fetching the texture coordinates from TU 3. The texture coordinates of TU 3 code the view direction on a per-vertex basis; therefore, during scan-conversion the interpolated view direction has to be re-normalized, either by a normalization cubemap or in the fragment program. For the sake of simplicity, we chose latter variant. In step 4 the layer color, fetched from the color palette in TU 2, is weighted by the calculated intensity c_{new}. Step 5 accomplishes the blending of the separate layer colors with a special attention paid to avoiding blending with the background color. In our implementation, the blending ratio of all layers is equal. Step 6 handles the intrinsic generation and handoff of the n-th layer lighting information. If so far the n-th layer has not been reached or outstepped by the current fragment, then the color in TU 1 is propagated as the intermediate result. Otherwise, the lit fragment is handed over to the subsequent slices. It is important to remark, that step 6 must not contain any kills, since the active layer information cannot be transferred to the next slice and is therefore lost.

Note, that in our implementation we use front-to-back blending of the single layers. The blending order might not matter, if we blend the layers equally. If we assign opacities for to the single layers manually, e.g in TU 2, then correct blending of the layers is achieved by the **under**, rather than by the **over** operator [19].

4. Applications in NPR

In this section we briefly explain, how the technique above can be used to distinguish between visible and hidden silhouettes of a volumetric object, as used in polygonal models, e.g. in [11]. For the sake of simplicity, we are going to depict hidden silhouettes of increasing layer number with brighter grey colors, which conforms with the expectations of the human perception of sensing depth by atmospheric effects, like e.g. fog [20]. Stylization features, like dashed lines, are not focus of this work.

Our algorithm needs to be rearranged only slightly to meet above requirements. While keeping the rendering pipeline unchanged, the fragment program is varied according to figure 2. Steps 1 and 2 remain unchanged. Step 3 selects the silhouette color to be the layer color in TU 2, if the view vector is nearly orthogonal to the gradient. Otherwise, the background color is propagated to flag that a silhouette can not be found at the current window position for the current layer. Step 4 keeps the current color in the framebuffer unchanged, if already a silhouette has been encountered for previous, lower layers. Step 5 assures, that only the first n layers contribute to the final image.

5. Results

The first two rows in figure 3 show (from left to right) the first three layers of the head and engine datasets and

```
For each fragment of the current slice:

  1. Determine $\alpha$-test.
     sign $\leftarrow (-1)^{layer\#+1}$

  2. $\alpha$-test.
     $sign \cdot \alpha < sign \cdot \alpha_{iso}$ ?
     Yes: Kill fragment.
     No: layer $\leftarrow$ layer# + 1;

  3. Coloring.
     $c_{new} \leftarrow \begin{cases} c_{layer} & : \quad |\overrightarrow{v_f} \cdot \overrightarrow{g_f} - 1| < \varepsilon \\ c_{back} & : \quad |\overrightarrow{v_f} \cdot \overrightarrow{g_f} - 1| \geq \varepsilon \end{cases}$

  4. Blending.
     $c_{new} \leftarrow \begin{cases} c_{old} & : \quad c_{old} \neq c_{back} \\ c_{new} & : \quad c_{old} = c_{back} \end{cases}$

  5. Layer test.
     $c_{result} \leftarrow \begin{cases} \boxed{c_{old}}\boxed{\text{layer}} & : \quad layer > n \\ \boxed{c_{new}}\boxed{\text{layer}} & : \quad layer \leq n \end{cases}$
```

Figure 2. *Survey of the fragment program for extraction of visible and hidden silhouettes.* We use the same terminology as in figure 1.

their composed result, respectively, blended with a ratio of $\frac{1}{3}$, for fixed α-thresholds. For the head, the outer and inner surface of the skull, and the brain is extracted. At the engine, the inner layers show peculiarities of the nozzles and tubes. The same datasets are used in the third and fourth row, this time visualized using the silhouettes described in section 4. From left to right, we show the visible silhouette and afterwards the three hidden silhouettes corresponding to the three subsequent hidden layers, and their superimposed result, respectively.

The implementation of our volume renderer is based on C/C++ using OpenGL and GLUT. The performance designations of table 2 were measured on a Windows 2000 PC with 768 MByte RAM, 1.4 GHz Intel P4 CPU and an ATI Radeon 9700 graphics card.

6. Conclusion

We presented a novel approach for extracting inner and outer boundaries of volumetric objects by introducing the concept of volumetric depth-peeling. The proposed approach is able to count the number of penetrations of a ray, casted into the scene through inner and outer iso-surfaces. Latter aim could only be achieved for two penetrations or for multiple rendering passes so far.

The user is able to explore the dataset in a fast and intuitive manner by only adjusting the iso-value and selecting the desired layer. Our method requires only a single rendering pass, independently of the number of layers extracted. We showed an application of the method in the field of NPR, where we are able to illustrate inner and outer silhouettes of the inspected object with colors of choice on a per-layer basis in a single pass. The proposed approach helps to understand the interior anatomy of datasets without the difficult user requirements for managing good transfer functions.

7. Acknowledgements

This work has been funded by the German Israeli Foundation (GIF)- Grant. Special thanks to Stefan Guthe for proofreading the paper, as well as for our reviewers for important hints for improving our work.

References

[1] F. Crow. Transparency for computer synthesized images. *Computer Graphics (Proceedings of SIGGRAPH 77)*, 11:242–248, July 1977.

[2] B. Csébfalvi and E. Gröller. Interactive volume rendering based on a "bubble model". *Graphics Interface*, 2001.

[3] P. Diefenbach. Pipeline rendering: Interaction and realism through hardware-based multi-passrendering. *Ph.D. dissertation*, 1996. University of Pennsylvania, Department of Computer Science.

[4] J. Diepstraten, D. Weiskopf, and T. Ertl. Transparency in interactive technical illustrations. In *Computer Graphics Forum*, volume 21, 2002.

[5] C. Everitt. Interactive order-independent transparency. *White paper, NVidia*, 1999. A Specification (Version 1.2.1). Silicon Graphics.

[6] H. Hauser, L. Mroz, G.-I. Bischi, and M. Gröller. Two-level volume rendering- fusing mip and dvr. *IEEE Visualization 2000 Proceedings*, pages 211–218, 2000.

[7] T. Heidmann. Real shadows, real time. In *Iris Universe, No. 18*, pages 23–31. Silicon Graphics Inc., Nov. 1991.

[8] V. Interrante, H. Fuchs, and S. M. Pizer. Enhancing transparent skin surfaces with ridge and valley lines. In *IEEE Visualization 1995 Proceedings*, pages 52–59, 1995.

[9] V. Interrante, H. Fuchs, and S. M. Pizer. Illustrating transparent surfaces with curvature-directed strokes. In *IEEE Visualization 1996 Proceedings*, pages 211–218, 1996.

[10] V. Interrante, H. Fuchs, and S. M. Pizer. Conveying the 3d shape of smoothly curving transparent surfaces via texture. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):98–117, 1997.

Window size	256^2	512^2
Head	6.66	1.88
Engine	8.73	2.80

Table 2. *Performance measurements in fps***.**

[11] R. Kalnins, P. Davidson, L. Markosian, and A. Finkelstein. Coherent stylized silhouettes. *Computer Graphics (Proceedings of SIGGRAPH 03)*, 22, 2003.

[12] D. Kay and D. Greenberg. Transparency for computer synthesized images. *Computer Graphics (Proceedings of SIGGRAPH 79)*, 13:158–164, 1979.

[13] J. Kniss, G. Kindlmann, and C. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):270–285, 2002.

[14] A. Lu, C. Morris, D. Ebert, P. Rheingans, and C. Hansen. Non-photorealistic volume rendering using stippling techniques. *IEEE Visualization 2002 Proceedings*, pages 211–218, 2002.

[15] A. Lu, C. Morris, J. Taylor, D. Ebert, P. Rheingans, C. Hansen, and M. Hartner. Illustrative interactive stipple rendering. *IEEE Transactions on Visualization and Computer Graphics, Vol. 9, No. 2*, 2003.

[16] A. Lu, J. Taylor, M. Hartner, , D. Ebert, and C. Hansen. Hardware-accelerated interactive illustrative stipple drawing of polygonal objects. *VMV 2002: Vision, Modeling and Visualization*, 2002.

[17] E. Lum and K.-L. Ma. Hardware-accelerated parallel nonphotorealistic volume rendering. *International Symposium on Nonphotorealistic Rendering and Animation (NPAR 02')*, 2002.

[18] A. Mammen. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *IEEE Computer Graphics and Applications, 9(4)*, pages 43–55, July 1989.

[19] K. Mueller and R. Crawfis. Eliminating popping artifacts in sheet buffer-based splatting. *IEEE Visualization 1998 Proceedings*, pages 239–245, 1998.

[20] T. Strothotte and S. Schlechtweg. *Non-Photorealistic Computer Graphics. Modeling, Rendering and Animation.* Morgan Kaufman Publishers., 2002.

[21] D. Weiskopf, K. Engel, and T. Ertl. Volume clipping via per-fragment operations in texture-based volume visualization. *IEEE Visualization 2002 Proceedings*, pages 93–100, 2002.

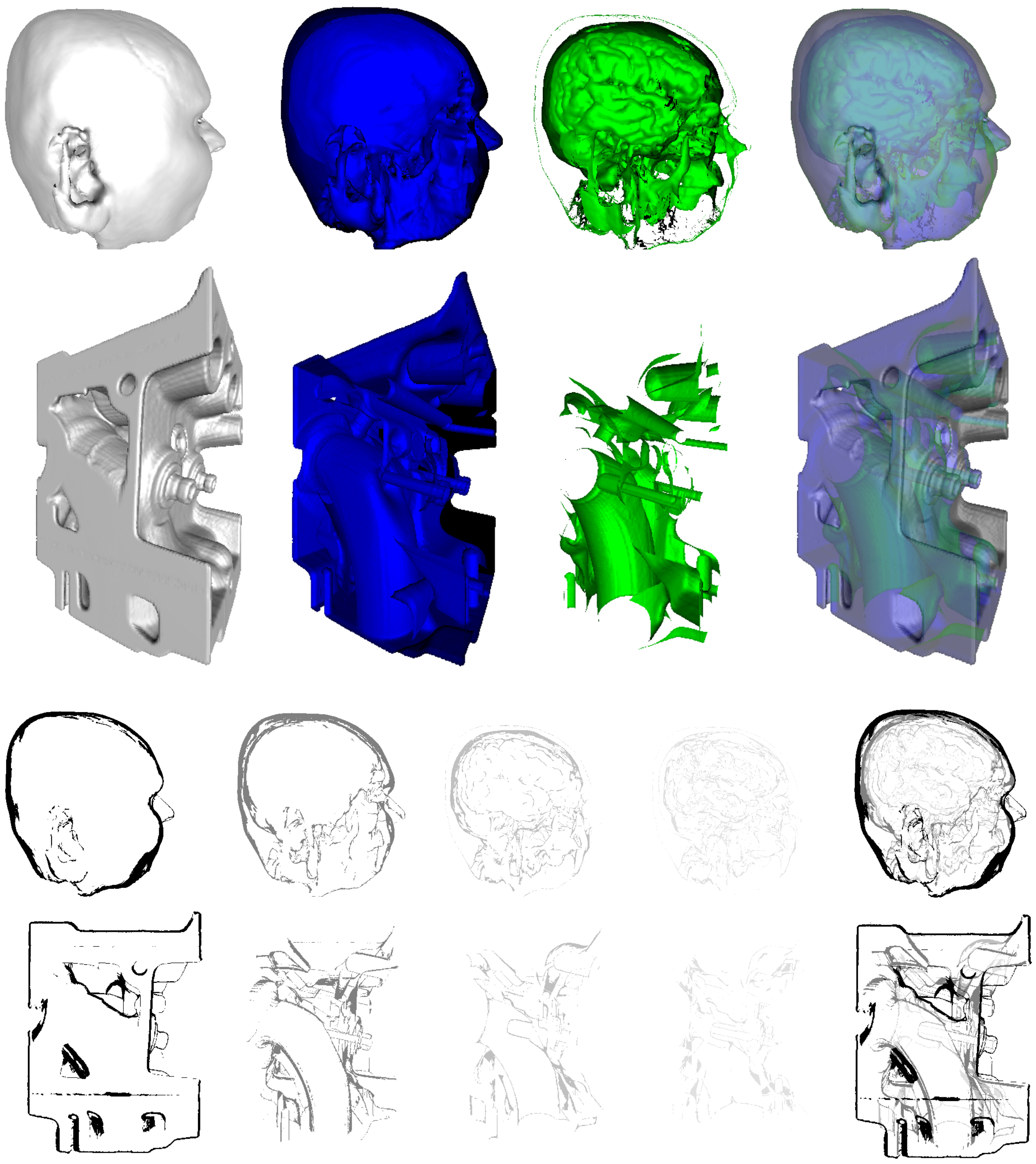

Figure 3. *Rendering results***.** The first two rows illustrate the depth-peeling method, applied to blend layers in equal ratios. The blended results for the head and the engine datasets are shown on the right, respectively. The third and fourth row show the same datasets, this time visualized with the NPR technique of section 4. The pictures on the right show the combined results, respectively. See text for details.