

Direct Volume Rendering for Polygon Models Embedded into Volumetric CT Data

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Marko Mlinaric

Matrikelnummer 0825603

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dr. Eduard Gröller
Mitwirkung: Dipl.Ing. Gabriel Mistelbauer

Wien, 29.11.2013



(Unterschrift Verfasser)

(Unterschrift Betreuer)

Direct Volume Rendering for Polygon Models Embedded into Volumetric CT Data

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Medical Informatics

by

Marko Mlinaric

Registration Number 0825603

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dr. Eduard Gröller

Assistance: Dipl.Ing. Gabriel Mistelbauer

Vienna, 29.11.2013



(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Marko Mlinaric
Piaristengasse 9/1/20, 1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29.11.2013

(Ort, Datum)

Mlinaric Marko

(Unterschrift Verfasser)

Acknowledgements

I would like to thank Gabriel Mistelbauer for his assistance and guidance with this thesis and the implementation. Also, I would like to thank the developers of the AngioVis Toolbox for letting me use their software for this work.

Kurzfassung

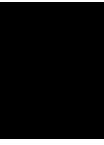
Für eine bessere Darstellung und ein besseres Verstehen von volumetrischen Daten kann die Kombination aus Surface Rendering (dem Rendern von Oberflächen) und Volume Rendering (dem Rendern von volumetrischen Daten) genutzt werden. Da sich die Technik zum Darstellen von Oberflächen aus Polygonen grundsätzlich von jener zur Darstellung von Volumen unterscheidet, ist eine Kombination nicht trivial. Die präsentierte Methodik erweitert die fachüblichen Techniken *Maximum Intensity Projection (MIP)*, *Direct Volume Rendering (DVR)* und *Maximum Intensity Difference Accumulation (MIDA)* um Mittel zur Darstellung von Oberflächen. Dies wird durch das Speichern von Positionen, Normalen und Farben des Polygon-Modells in einen Buffer und anschließender Verwertung im Ray Marching Algorithmus realisiert. Die Methodik wurde als Plugin für die AngioVis Toolbox implementiert.

Abstract

For a better visualization and understanding of volumetric data, combinations of surface and volume rendering can be used. Since, the rendering techniques used for polygon surfaces and volumes are different, the combination is not trivial. The presented methodology enhances the state-of-the-art techniques *Maximum Intensity Projection (MIP)*, *Direct Volume Rendering (DVR)* and *Maximum Intensity Difference Accumulation (MIDA)* with methods to render surfaces, by buffering the positions, normals and colors of the polygon models, and utilizing them in the ray marching process. The methods have been implemented in the AngioVis Toolbox as a plugin.

Contents

1	Introduction	1
2	Related Work	3
2.1	Volume Rendering	3
2.2	Visualization of Vascular Structures	4
2.3	Combined Techniques	4
3	Methodology	7
3.1	Basic Volume Rendering	7
3.2	Slice Plane Rendering	9
3.3	Vessel Tree Rendering	12
3.4	Transparent Plane	14
4	Implementation	17
4.1	AngioVis	17
4.2	OpenGL	17
4.3	CUDA	17
5	Results	19
5.1	Result Images	20
6	Conclusion	25
	Bibliography	27



Introduction

The aim of medical visualization is to enhance diagnosis, examination and exploration of the human body by illustrating radiological data. Data is acquired using technologies like Magnetic Resonance Imaging (MRI) and Computed Tomography (CT) and usually represented in a volume, which is a 3D data set. In order to display an overview of this data, there exist several techniques like Maximum Intensity Projection (MIP) and Direct Volume Rendering (DVR).

For a better visualization and understanding of the volumetric data, parts of the body, e.g. blood vessels or bones, can be modeled and rendered as surfaces. Surfaces can be described as a polygon mesh or defined as an implicit function. Inserting this surface model into a translucent volume can aid to a better quality of the resulting image. However, since volumetric data and surfaces are usually rendered by different techniques, a combination of them is not trivial. The aim of this work is to implement and present such a technique by rendering models of blood vessels inserted into a CT data set.

In Chapter 2, related work will be presented, showing techniques used to render volumes, surfaces or combinations of them. In Chapter 3, our method will be presented in three steps: Rendering a slice plane, rendering a polygon model of blood vessels and rendering a transparent plane. In Chapter 4, we will introduce the used frameworks and APIs and show implementation details. Results will be represented in Chapter 5. Lastly, Chapter 6 contains conclusion and opportunities for future work.

Related Work

In this chapter, we will present related work, showcasing the state of the art in volume rendering, visualization of vascular structures and combined techniques.

2.1 Volume Rendering

In Volume Ray Casting, for each pixel on the image plane a ray is shot from the point of view through the volume. While this ray traverses the volume, values are sampled from the 3D data set. A transfer function is used to calculate color and opacity from the sampled value (e.g. density). It depends on the used technique, how the color from the ray is used for the according pixel.

Maximum Intensity Projection (MIP) was described by Wallis et al. [9] and is thanks to its simplicity and speed a commonly used approach. For each ray, the maximum value on that ray is determined and drawn to the pixel. It is a computational simple, thus fast technique, however, it does not provide a good sense of depth on still images. Only by rotating the volume a better sense of depth can be gained.

Levoy [5] described a rendering pipeline for Direct Volume Rendering (DVR). While the ray traverses the volume, color and opacity are saved, starting with color $(r, g, b) = (0, 0, 0)$ and opacity $\alpha = 0$. For each step color C_i and opacity A_i are accumulated with a transparency function.

$$\begin{aligned} C_i &= C_{i+1} + (1 - A_{i+1})C_i \\ A_i &= A_{i+1} + (1 - A_{i+1})A_i \end{aligned}$$

The advantage of this technique is a better sense of depth. However, because of the accumulation, for the viewer important parts of the volume can be hidden by overlapping objects, when using a suboptimal transfer function.

Bruckner et al. [3] described an approach, which had the aim to combine MIP and DVR. On the one hand, the advantage of MIP is, that it renders the parts of the volume with a high intensity, while on the other hand, DVR provides a better sense of depth. MIDA should reduce the necessity of finding a good transfer function in DVR.

In order to combine mentioned advantages, in addition to the ray marching algorithm described for DVR, the maximum intensity on the ray to the current step f_{max} is saved, similar to MIP. d_i is used to classify the change for each step, being calculated by

$$d_i = \begin{cases} f_i - f_{max} & \text{if } f_i > f_{max} \\ 0 & \text{otherwise.} \end{cases}$$

The weighting factor b_i is then calculated by $b_i = 1 - d_i$. Finally, the transparency function of DVR is adjusted by the weighting factor.

$$\begin{aligned} C_i &= b_i C_{i+1} + (1 - b_i A_{i+1}) C_i \\ A_i &= b_i A_{i+1} + (1 - b_i A_{i+1}) A_i \end{aligned}$$

2.2 Visualization of Vascular Structures

Preim et al. [6] gave an overview of techniques used for visualization of vascular structures. They discussed approaches using Direct Volume Rendering or Surface Rendering and usage of model assumptions, as well as illustrative visualization of vasculature and interactive exploration of the data.

Convolution surfaces were presented by Bloomenthal et al. [2] and describe methods to deduce the surface from the skeleton. By this, smooth surfaces are generated and changes on the underlying skeleton structure cause changes of the surface.

Wu et al. [10] presented an approach for visualization of vascular structures based on their centerline and radius. Smooth transitions were achieved by bidirectional sampling and curvature-dependent subdivision was used to achieve a better trade-off between mesh size and mesh quality.

2.3 Combined Techniques

Dizhevskii et al. [4] described methods for rendering 3D objects inserted into a transparent volume. They offer methods for rendering implicit surfaces or polygon models. Considering their approach using surfaces represented by polygons, they represent two possible solutions. The first is the voxelization of the polygons, which can be understood as a 3D rasterizer, where the surface properties are added to the volume data and considered in their ray marching algorithm. The other method describes a 2D hashing algorithm, where the idea is to project each polygon to cells on the image plane, thus reducing the amount of polygons, for whom intersection with the ray traversing through the volume has to be calculated.

A similar approach is presented by Ruijters et al. [7]. In their work, they represent at first a method for registration of blood vessel to volumetric data. Second, a method is described, where the surface data is rendered in a first pass to a depth buffer. The volume rendering algorithm is then extended by a per-step check to the depth buffer, determining if the ray has passed the surface. Lastly, they offer a method for rendering vessel silhouettes and discuss clinical use of the method.

Tietjen et al. [8] described methods for combination of volume, surface and line rendering using a scene graph based approach, mainly for medical use. They presented an object-based edge-detection used for generation of silhouettes.

Another approach was presented by Bhalerao et al. [1], introducing a super-z depth buffer, storing color and opacity to a given approximation, while assuming a fixed viewpoint. The method was developed for fast re-rendering, after changes in location, color or opacity occur.

Methodology

In this chapter, we will present our methodology. In the first section, we will describe Volume Rendering without polygon surfaces. Next, we will add an opaque plane to display the current axial slice position within the volumetric data. The third section describes, how we enhanced our method to add color and normals to the vertices using the example of a model of blood vessels. Lastly, we will enhance the plane in order to be transparent.

The naive approach would be to render the surface above the volume. Comparisons between the naive approach and the following method are presented later, in Chapter 5.

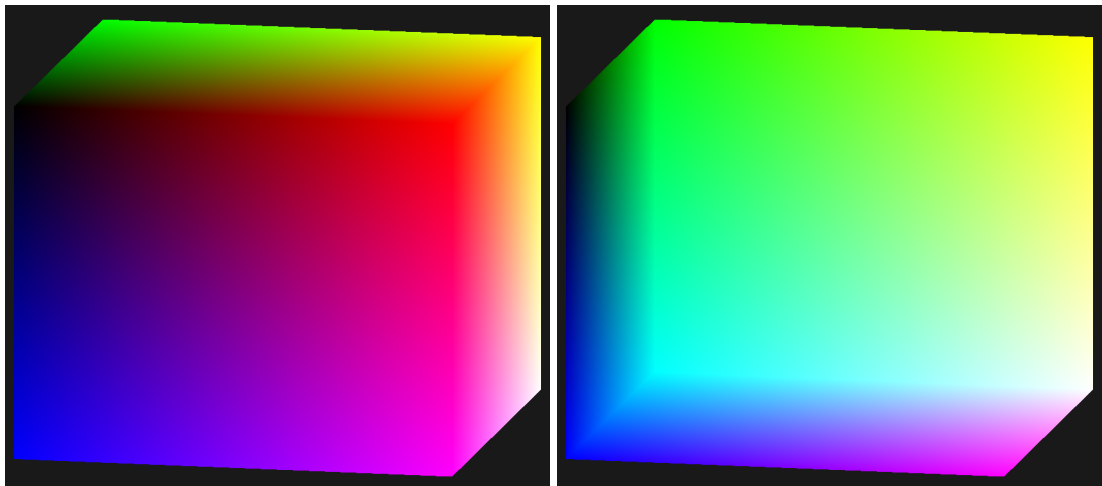
3.1 Basic Volume Rendering

For the ray marching algorithm, as it was described in Section 2.1, the direction of the ray and intersection with the volume are necessary. They can be gathered by determining the entry and exit points of the ray to the volume. Afterwards, the ray directions can be calculated as the vector pointing from the front side to the back side, for each pixel. In the ray marching loop, the position of the ray for each loop step is calculated by adding a fraction of the ray to the last position, beginning with the entry point and ending with the exit point. The fraction is determined by the step size or step length respectively. If there are no entry or exit points, than there is no volume hit and the background color is drawn to the pixel.

In order to get the entry and exit points to the volume, first a bounding box for the volume is created. We assume that the volume is in the first octant, has its minimum at $(0, 0, 0)$ and its maximum at $(1, 1, 1)$. This way, each position value of the volume can be saved to a color, while setting the alpha value to 1.

Two render passes are performed. The first renders the front side of the bounding box to one texture, the second renders the back side to another.¹ The color on the faces is interpolated accordingly. As a result, the textures display the two sides of a colored cube, see Figure 3.1. Each pixel contains the entry or exit point respectively for each ray and is retrieved by a texture

¹In OpenGL, this can be achieved using Face Culling.



(a) Entry Points

(b) Exit Points



(c) MIP

Figure 3.1: The position of the entry points (a) and exit points (b) to the volume are mapped to a color and drawn to a texture each, this is used to determine the ray. Their orientation determines the orientation of the volume (c).

look up. Because a cleared texture has at each point the value 0, however, the alpha value of the bounding box was set to 1 at each position, this is used to determine a volume hit.

As a note for volumes, which do not have the assumed properties, their bounding box can be transformed accordingly by a transformation matrix. Later, the inverse matrix is used to calculate their original positions.

Our ray marching algorithm works according to the presented state of the art in Section 2.1 and was implemented for MIP, DVR and MIDA.

3.2 Slice Plane Rendering

The plane is used to display the position of a 2D slice image in the volume. It is located inside of the bounding box of the volume and has one by the user predefined color.

The positions of the plane are rendered to a texture, the same way it is done with the front and back sides of the volume bounding box, see Figure 3.2a. Since the plane is flat, it is sufficient to render the front side only. This way, intersections of the ray with the plane are gathered. According to determining a volume hit, a plane hit is determined at the points, where the alpha value of the texture is 1.

Following, we will show, how three major Volume Rendering techniques have to be enhanced, in order to render the plane.

3.2.1 MIP

Along the ray marching the maximum intensity is saved and later transferred to a color. As an enhancement the position of this point is also saved. This is used to determine, if the volume point with the maximum intensity is closer than the surface point on the plane.

If the point on the plane is closer, the plane color is rendered, since the plane is opaque. Otherwise, the color of the value is calculated by using a transfer function and blend over the plane color.

3.2.2 DVR and MIDA

For DVR, first, the step of the ray marching loop, where the ray will intersect the plane, is calculated. At this point of the loop, the plane color is accumulated. The loop can be terminated at this point, as the plane opacity is 1 and no values behind it would be visible (see Listing 3.1).

The loop for MIDA is gathered, by exchanging the `accumulate()` function of DVR and saving the maximum opacity, as it was shown in Section 2.1.

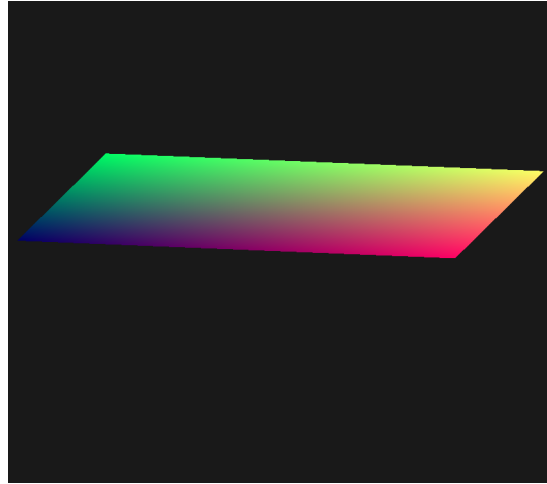
Listing 3.1: Entry, exit and plane surface points are retrieved by texture look ups and used to calculate ray and intersection point. Inside of the ray marching loop, for each step it is checked, if the intersections occurred and, if so, the color of the plane is accumulated and the loop breaks. If not, conventional accumulation according to DVR is used.

```
1 // Length of ray to surface point on plane
2 float surface_depth = length(surface_point - entry_point);
3 // Length of ray in volume
```

```

4 float total_depth = length(exit_point - entry_point);
5 // Step of ray marching loop, where ray will intersect plane
6 int intersection_step = (surface_depth / total_depth) * step_count;
7
8 // Ray marching loop
9 for (int i = 0; i < step_count; i++) {
10 // Check for intersection with plane
11 if (i == intersection_step) {
12 // Plane color is rendered
13 result = accumulate(result, plane_color);
14 // Break, since plane is opaque
15 break;
16 }
17 // Determine current position of ray
18 float3 position = entry_point + i * step_length * ray_direction;
19 // Retrieve sample using trilinear interpolation
20 float sample = volume_data.trilinear(position);
21 // Use transfer function
22 float4 sample_color = transfer(sample);
23 // Accumulate transfered color
24 result = accumulate(result, sample_color);
25 }

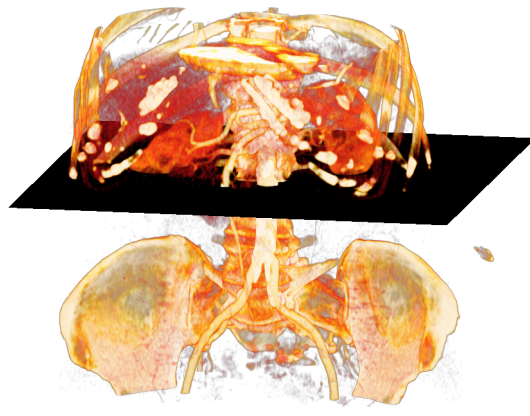
```



(a) Slice Positions



(b) MIP



(c) DVR

Figure 3.2: The position of the slice plane are mapped to a color and drawn to a texture (a). This is used to determine the ray intersection and applied for MIP (b), DVR (c) and MIDA.

3.3 Vessel Tree Rendering

The polygon model of the blood vessels can be rendered the same way as the plane. However, the model is curved and thus it would aid to the visibility, if it were shaded. In order to do this, it is necessary to determine the normals. Additionally, we assumed that the plane only has one color. This does not have to be the case for other polygon models, therefore, it is also necessary to determine the color on each surface point.

Three additional textures are used. The first holds the positions of the model, as it is done for the plane and the front faces of the bounding box of the volume previously. See Figure 3.3a.

A second pass is used to render the colors of the model to the second texture. This is basically the normal rendering of the unshaded model.

The third pass renders the normals to the third texture. However, since normals can have negative values, it is necessary to transform each coordinate from the interval $[-1, 1]$ to the interval $[0, 1]$. Afterwards, in the ray marching algorithm, they have to be transformed back, in order to shade the vessel tree properly. See Figure 3.3b.

Positions, normals and colors are retrieved by a texture look up at ray coordinates.

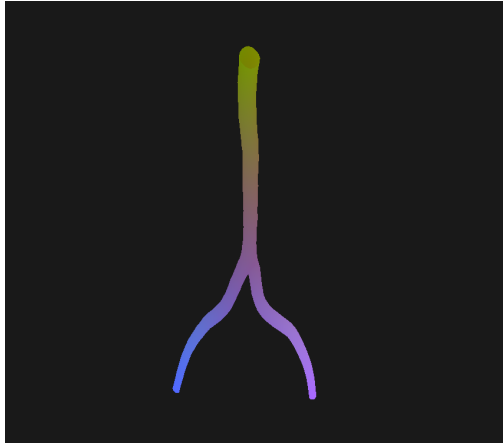
3.3.1 MIP

The method described in Section 3.2.1 has to be enhanced to support multiple surfaces and shading. For this, the closest point P of a surface on the ray has to be determined. P is then shaded, unless it is a point on the plane.

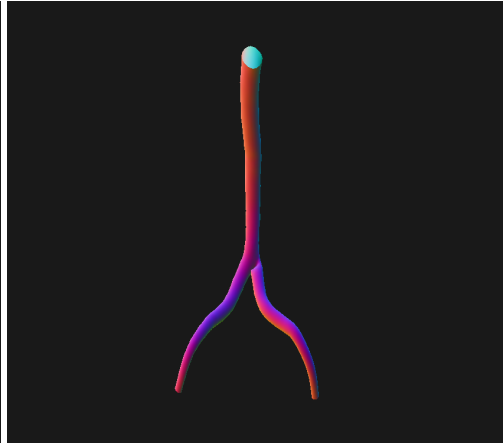
If P is closer than the point in the volume, then the shaded surface color is rendered to the pixel. Otherwise, the color of the value is calculated by using a transfer function and blend over the shaded surface color.

3.3.2 DVR and MIDA

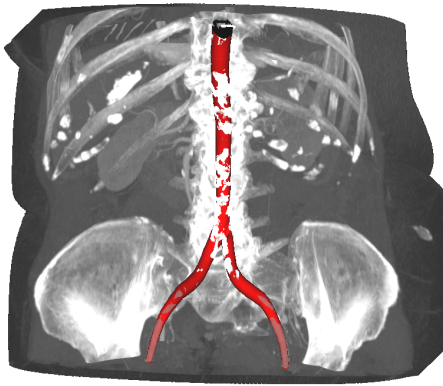
The method described in Section 3.2.2 is enhanced by calculating the intersection for each surface point with the ray. For the closest of these points, the step of the ray marching loop, where intersection occurs, is calculated. Additionally, the point is shaded and the shaded color is used at the intersection step.



(a) Vessel Positions



(b) Vessel Normals



(c) MIP with vessels



(d) MIDA with vessels

Figure 3.3: The positions (a) and normals (b) to the vessel structure are mapped to a color and drawn to a texture each. This is used to determine the ray intersection and shaded color and applied for MIP (c), MIDA (d) and DVR.

3.4 Transparent Plane

Until now, we have shown, how opaque objects can be rendered. Now, we will extend this to simple transparent objects and show it by making the slice plane transparent. For this, further enhancements on the used render techniques are necessary.

Note, that following render methods can handle multiple surfaces, as long as the transparent surfaces are not self occluding. It is only necessary to provide multiple textures, which contain positions, normals and colors for each surface model.

3.4.1 MIP

Since an object can be transparent, it is necessary to sort all points, which intersect the ray. In the shown examples, these are the transparent volume color, the transparent plane color and the opaque vessel color. In a loop over the sorted points, beginning with the closest, the points are accumulated, similar to DVR. If the last point was opaque, the loop can be terminated.

Listing 3.2: For MIP, all surfaces are sorted according to their depth. Afterwards, they are accumulated, similar to DVR.

```
1 // Get colors of sorted points
2 float4 colors[] = ...
3 // Set first color to current result
4 result = colors[0];
5
6 // Loop over all values
7 for(int i=1; i<colors.length; i++) {
8     if (result.alpha<1.0) {
9         // Break loop, because opaque surface has been accumulated
10        break;
11    }
12
13    // Accumulate color
14    result = accumulate(result, colors[i]);
15 }
```

3.4.2 DVR and MIDA

In the ray marching algorithm it is necessary to check for intersections with all surfaces. This is done by calculating for each point the steps of the loop, at which intersection with the ray will occur. Inside of the loop, all steps are checked for intersection.

As before, the only differences of MIDA to DVR is, that the maximum opacity has to be saved and the adjusted accumulation function has to be used.

Listing 3.3: For DVR, the intersection step of each surface is calculated. Afterwards, for each step intersection is checked.

```
1 // Calculate array with intersection steps and corresponding colors
2 int intersection_steps[] = ...
3 float4 colors[] = ...
```



```

4
5 // Ray marching loop
6 for (int i = 0; i < step_count; i++) {
7     // Loop over all intersections
8     for (int j = 0; j < intersection_steps.length; j++) {
9         // Check if is intersection step
10        if (i == intersection_steps[j]) {
11            // Accumulate corresponding surface color
12            result = accumulate(result, colors[j]);
13        }
14    }
15    // Determine current position of ray
16    float3 position = entry_point + i * step_length * ray_direction;
17    // Retrieve sample using trilinear interpolation
18    float sample = volume_data.trilinear(position);
19    // Use transfer function
20    float4 sample_color = transfer(sample);
21    // Accumulate transfered color
22    result = accumulate(result, sample_color);
23 }

```

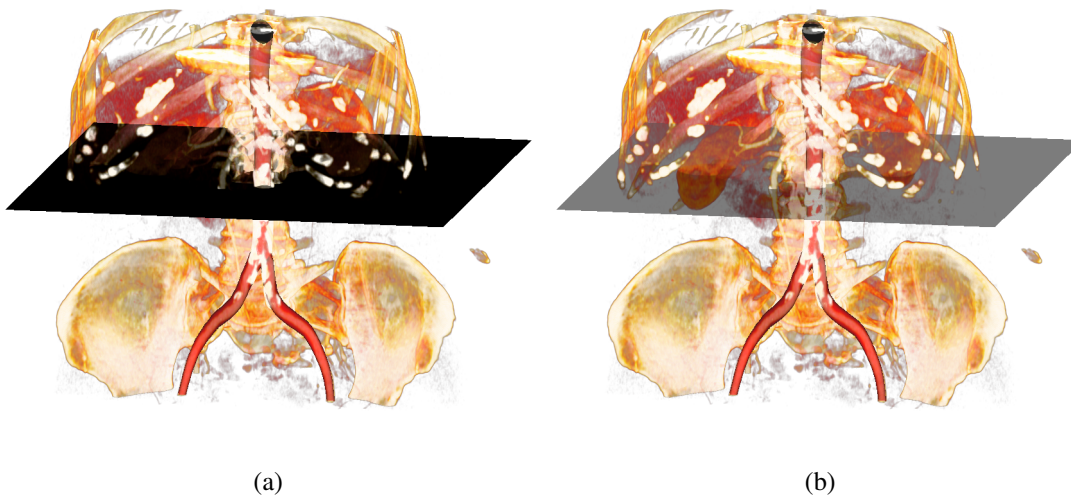


Figure 3.4: MIDA with opaque plane (a) and 50% transparent plane (b)

Implementation

4.1 AngioVis

AngioVis is a project that deals with visualization and processing of CTA datasets. Processing of these datasets to identify blood vessels is performed by usage of semi-automatic algorithms. In addition, several methods for visualization of the results are provided.

The methods presented in this work have been implemented as a plugin into the framework. As a plugin, the implemented methods had access to loaded datasets and already generated blood vessel structures.

4.2 OpenGL

OpenGL is an API used to interact with Graphics Processing Units (GPUs) for rendering 2D and 3D computer graphics and is managed by the consortium Khronos Group. For our implementation we used OpenGL 3.2 compatibility profile.

For off-screen rendering OpenGL offers frame buffer objects (FBOs) as an extension. These are suitable for rendering a scene to a texture. For our implementation, this was used to render the polygon models to textures.

4.3 CUDA

The Compute Unified Device Architecture (CUDA) is a general purpose parallel computing platform developed by NVidia for their GPUs. CUDA enables developers to outsource computational tasks directly on the GPU and process data in a parallel way. Since volume rendering can be parallelized well, because each ray can be processed independently, CUDA is a convenient tool for the aim of this work.

Parallel functions are declared using the keyword `__global__`. They are called with `parallel_function<<<GRIDSIZE, BLOCKSIZE>>>()`. In context to this work, the

result image is covered by a grid consisting of blocks. BLOCKSIZE describes the dimension in pixels of a cell on this grid, thus GRIDSIZE being the amount of blocks in x- and y-direction. The area of a block determines the amount of threads running parallel on the GPU. Each thread corresponds to a pixel on the image. The calculation of the coordinates is shown below.

```
1  uint x = blockIdx.x * blockDim.x + threadIdx.x;
2  uint y = blockIdx.y * blockDim.y + threadIdx.y;
3
4  uint offset = y * width + x;
```

In order to use OpenGL textures in parallel functions, it is necessary to map them to CUDA. First, memory on the GPU is allocated and the OpenGL texture is registered to CUDA as a resource. Then this resource is mapped to a CUDA array and finally copied to that array. This process is shown in the code snippet below. The pointer to the device memory is passed as a parameter to the parallel function.

Listing 4.1: CUDA Memory Mapping

```
1  // allocate device memory and register texture as CUDA resource
2  cudaMalloc(&devMemPtr, sizeof(float4)*width*height);
3  cudaGraphicsGLRegisterImage(&cudaResource,
4      textureID, GL_TEXTURE_2D, cudaGraphicsRegisterFlagsNone);
5
6  // map resource to array
7  cudaGraphicsMapResources(1, &cudaResource, 0);
8  cudaGraphicsSubResourceGetMappedArray(&cudaArray, cudaResource, 0, 0);
9
10 // copy memory
11 cudaMemcpyFromArray(devMemPtr, cudaArray, 0, 0,
12     sizeof(float4)*width*height, cudaMemcpyDeviceToDevice);
```

Results

In this chapter, we will show several result images of our implementation. MIP, MIDA and DVR will be compared with each other and also comparisons to the naive approach will be shown. For this, we will show typical results on two selected data sets.

Dataset	Dimensions
A	512 x 512 x 256
B	512 x 512 x 401

Table 5.1: Used Datasets

Following, it can be seen that in comparison to the naive approach a better sense of depth can be achieved. Especially in the case of MIP, which usually provides inferior reception of depth, a plane can aid orientation. However, since points, which are adjacent in the result image, can be quite apart from each other in the volume, there is some noise on the region, where the surface intersects a dense part of the volume.

MIDA is well suited to accentuate the surfaces inside of the volume. This can be seen in Figure 5.3a. Additionally, in Figure 5.4a, it is visible, that the problem of DVR with covered parts of the volume transfers to the combination with surface rendering.

For MIP, we used the parameters $C = 1300$ and $W = 1300$ for the windowing function. For DVR and MIDA, $C = 1600$ and $W = 1000$ was used.

5.1 Result Images



(a) MIP

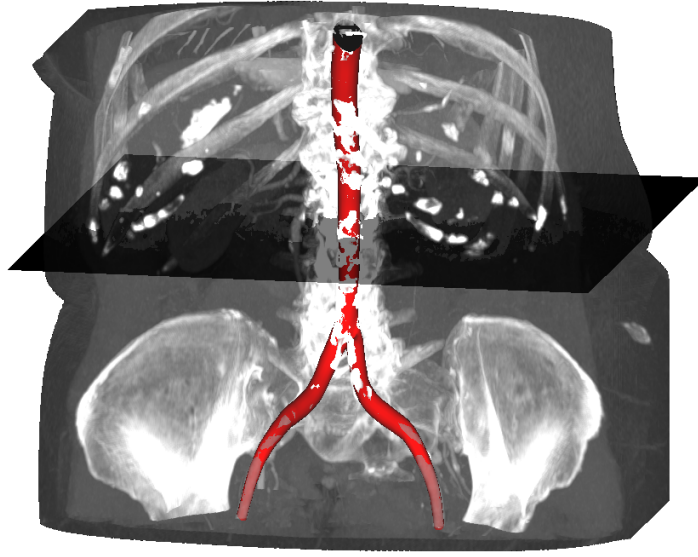


(b) MIDA

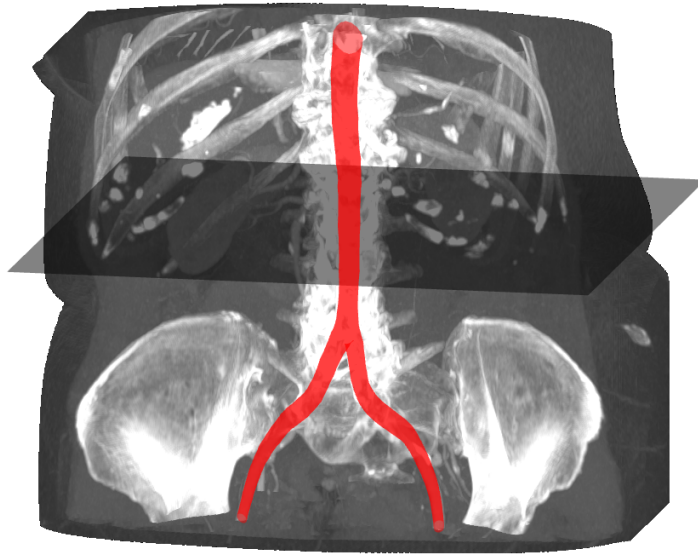


(c) DVR

Figure 5.1: Comparison of MIP (a), MIDA (b) and DVR (c) in Dataset B, plane transparency 50%.

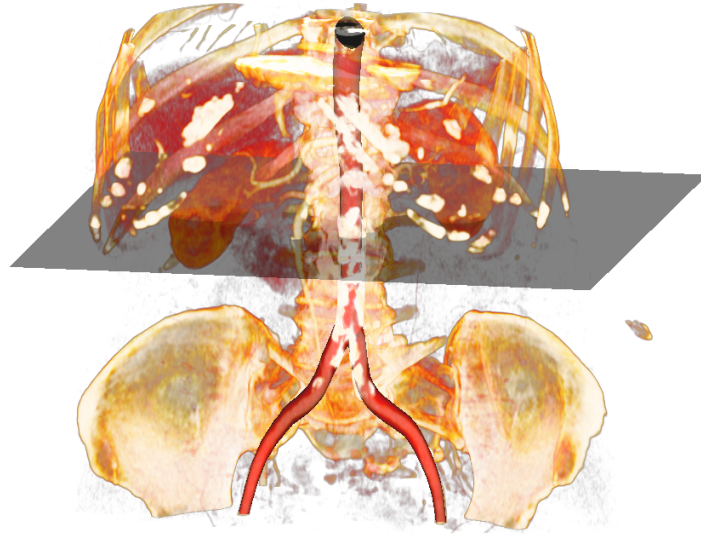


(a)

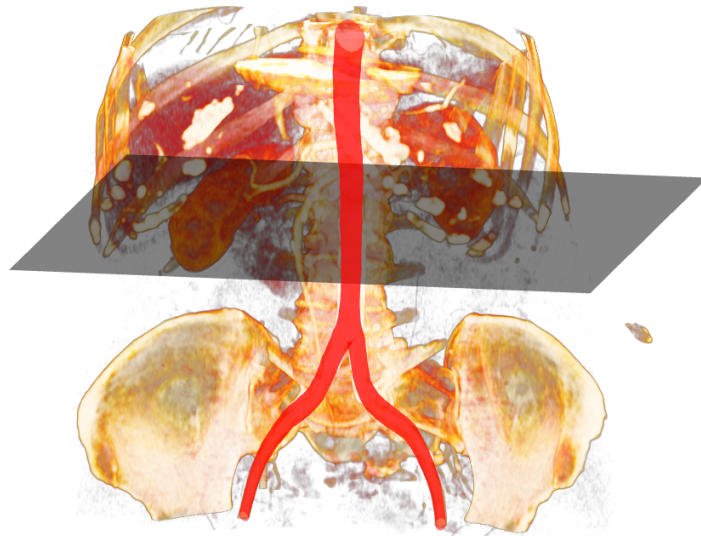


(b)

Figure 5.2: Comparison of our (a) and the naive (b) approach for MIP in Dataset A, plane transparency 50%.

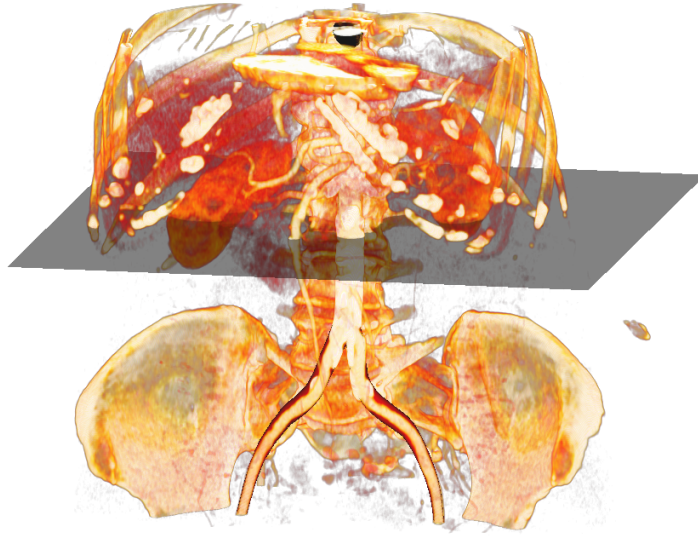


(a)

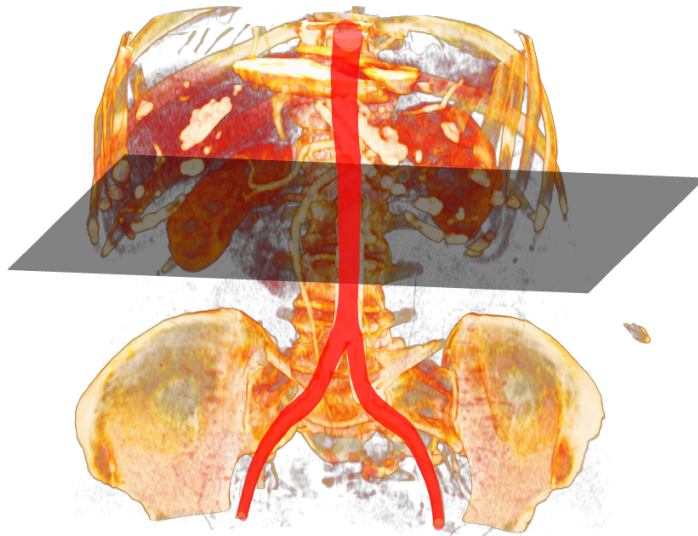


(b)

Figure 5.3: Comparison of our (a) and the naive (b) approach for MIDA in Dataset A, plane transparency 50%.



(a)



(b)

Figure 5.4: Comparison of our (a) and the naive (b) approach for DVR in Dataset A, plane transparency 50%.

Conclusion

We showed a method to combine surface and volume rendering by enhancing the state-of-the-art techniques *Maximum Intensity Projection (MIP)*, *Direct Volume Rendering (DVR)* and *Maximum Intensity Difference Accumulation (MIDA)* with methods to render surfaces in order to enhance the visual representation of volumetric data. We did this by buffering the positions, normals and colors of the polygon models. These buffers were used in the ray marching algorithm to determine the intersection point of the surface with the ray.

The combination of volume and surface rendering is a good method to enhance the visual appearance of volume data. It is shown, that important parts of the represented body can be highlighted.

There are **limitations** with transparency. Although they can be removed for some cases, rendering of transparent self-occluding objects is more complex.

Bibliography

- [1] A. Bhalerao, H. Pfister, M. Halle, and R. Kikinis. Fast re-rendering of volume and surface graphics by depth, color, and opacity buffering. *Journal of Medical Image Analysis*, 4:235–251, 09/2000 2000.
- [2] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In James J. Thomas, editor, *SIGGRAPH*, pages 251–256. ACM, 1991.
- [3] Stefan Bruckner and Meister Eduard Gröller. Instant volume visualization using maximum intensity difference accumulation. *Computer Graphics Forum*, 28(3):775–782, June 2009.
- [4] A.Yu. Dizhevskii. Rendering of 3d objects inserted into a translucent volume. *Moscow University Computational Mathematics and Cybernetics*, 32(4):227–233, 2008.
- [5] Marc Levoy. Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37, May 1988.
- [6] Bernhard Preim and Steffen Oeltze. *3D Visualization of Vasculature: An Overview*, chapter Visualization in Medicine and Life Sciences, pages 19–39. Springer Verlag, 2007.
- [7] D. Ruijters, D. Babic, B.M. ter Haar Romeny, and P. Suetens. Silhouette fusion of vascular and anatomical volume data. In *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on*, pages 121–124, 2006.
- [8] Christian Tietjen, Tobias Isenberg, and Bernhard Preim. Combining silhouettes, surface, and volume rendering for surgery education and planning. In *Proceedings of the Seventh Joint Eurographics / IEEE VGTC conference on Visualization*, EUROVIS’05, pages 303–310, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.
- [9] Jerold W. Wallis and Tom R. Miller. Three-dimensional display in nuclear medicine and radiology. *Journal of Nuclear Medicine*, 32(3):534–546, 1991.
- [10] Jianhuang Wu, Renhui Ma, Xin Ma, Fucang Jia, and Qingmao Hu. Curvature-dependent surface visualization of vascular structures. *Comp. Med. Imag. and Graph.*, 34(8):651–658, 2010.