Christof Sirk

# Dynamic Label Placement in Volumetric Scenes

**MASTER'S THESIS**

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Biomedical Engineering

submitted to

**Graz University of Technology**

Supervisor

Prof. Dr. Dieter Schmalstieg
Institute for Computer Graphics and Vision

Dr. Alexander Bornik
Ludwig Boltzmann Institute for Clinical-Forensic Imaging

Graz, Austria, February 2017

# Abstract

Presenting medical forensic scenes to laymen is a difficult task. To alleviate this challenge, annotations in the form of labels are added to the scene. We present a new algorithm to automatically place these labels at optimal positions in volumetric scenes as well as in scenes with many transparent objects.

To determine the position of a label, criteria for a good position and positional constraints are encoded in a cost function. To account for the third dimension, the scene is divided into layers parallel to the image plane. Each label is associated to a layer and the costs for each label in a layer are minimized sequentially.

A HA-Buffer, which is a kind of dictionary for fragments with intrinsic depth sorting, is used to support many transparent objects and provide an integration point for volume rendering. Volume rendering is performed using ray casting in a fragment shader.

The results show that the placed labels are comparable to state of the art implementations, which partially require manual input, but at a lower performance penalty. Due to the layered approach, additional depth cues make the scene more comprehensible.

Overall, the algorithm provides a good alternative to the state of the art labeling for normal 3D scenes and a novel combination with volumetric scenes.

**Keywords:** annotations, labeling, volume rendering, transparency

# Kurzfassung

Die Präsentation von medizinisch-forensischen Fällen für Laien stellt eine schwierige Aufgabe dar. Um diese zu vereinfachen, werden Annotationen in Form von Beschriftungen zur Szene hinzugefügt. Wir stellen einen neuen Algorithmus zur automatischen Positionierung dieser Beschriftungen in Szenen mit volumetrischen Daten und mit vielen transparenten Objekten vor.

Um die Position einer Beschriftung zu bestimmen, werden Kriterien und Bedingungen für eine gute Position in einer Kostenfunktion formuliert. Um die dritte Dimension zu berücksichtigen, wird die Szene in Schichten parallel zur Bildebene gegliedert. Jede Beschriftung wird einer Schicht zugeordnet und die Kosten der einzelnen Beschriftungen in einer Schicht werden sequenziell optimiert.

Ein HA-Buffer, eine Art Wörterbuch für Pixel mit einer intrinsischen Tiefensortierung, wird verwendet, um viele transparente Objekte darstellen zu können. Weiters wird dadurch die Integration mit dem Volume Rendering erleichtet. Volumen werden über Ray Casting — implementiert in einem Fragment Shader — dargestellt.

Die Ergebnisse zeigen, dass die Platzierung der Beschriftungen mit dem Stand der Technik vergleichbar ist, obwohl andere Algorithmen teilweise manuelle Eingaben erfordern. Die vorgestellte Implementierung ist dabei deutlich schneller als vergleichbare Algorithmen. Durch die Gliederung in Schichten sind weiters Tiefenhinweise zum besseren Verständnis möglich.

Allgemein bietet der Algorithmus eine gute Alternative zum Stand der Technik für normale 3D-Szenen und eine völlig neue Kombination für volumetrische Szenen.

**Schlüsselwörter** Annotationen, Beschriftungen, Volume Rendering, Transparenz

## Affidavit

*I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.*

*The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.*

| | |
|---|---|
| Date | Signature |

# Acknowledgments

First, I would like to thank my parents for providing me with the possibility to study at Graz University of Technology and my whole family for supporting me throughout the whole time.

I would especially like to thank DI Dr. Alexander Bornik of the Ludwig-Boltzmann-Institute for Clinical-Forensic Imaging (LBI) for posing this interesting topic and the many fruitful discussions during the whole project. Also, I would like to thank Prof. DI Dr. Dieter Schmalstieg of the Institute for Computer Graphics and Vision at Graz University of Technology (ICG) for supervising my thesis, even though the main work is for the LBI.

Many thanks to the colleagues at the LBI, especially DI Dr. Martin Urschler, who first brought the LBI to my attention and also for his support. My gratitude also goes to the ICG members, particularly DI Dr. Denis Kalkofen. He has given me invaluable advise derived from his long experience with label placement.

Moreover, I would like to thank my fellow students, with whom I had the pleasure to spend many hours, not only studying, but also enjoying life. In particular, I would like to thank Alexander Jesner, Josef Koller, Bernhard Rapp and Andreas Schwarzl for the close collaboration in the previous years and for taking the time to review my thesis and providing me with constructive feedback.

I am especially indebted to Martin Hatzl for creating beautiful illustrations for the thesis to help explain key aspects of the thesis and enduring my countless change requests.

Last but not least my gratitude goes to my superiors at my employer Anton Paar, who have enabled me to work part time. Also I would like to thank my colleagues for always having my back when I had to leave early or had to take a day off to study.

This work was partly supported by funding from the KIRAS program (grant no. 850183, CSISmartScan3D) under supervision of the Austrian Research Promotion Agency (FFG).

# Contents

# List of Figures

# List of Tables

# List of Listings

# 1 Introduction

Visualizing forensic medical data in a manner that is understandable to a layperson is a daunting task. In our case, the medical data is generated using computed tomography (CT), but it could also stem from magnetic resonance imaging (MRI) or other imaging modalities. A lot of forensic cases contain inner injuries, which are presented in court to judges, juries and lawyers. To guide the layperson to important medical findings, they are marked in the 3D visualization and annotated with labels. These are, for example, certain bone breaking patterns to infer the direction of an applied force during an accident. Therefore, the visualization tool is used to reconstruct crimes and accidents for analysis and for exporting videos and images for court presentations.

Placing labels at appropriate positions in a 2D or 3D environment is itself a difficult problem (NP-complete [30]). Combining this with volumetric data is even harder, because the problem cannot be tackled after rendering the scene, if the transparency should be incorporated into determining the label's position.

The goal of this work is to develop a labeling solution, that works well with volumetric data and many transparent objects. It should be an improvement of the existing implementation in the `VolyRenderer` software [52] currently in use, which has problems placing labels when no background is visible. Also, the rendering system should be able support the requirements of multiple volumes and many transparent objects.

We have integrated a labeling algorithm with a volume renderer and a rendering pipeline which is construed for numerous transparent objects. The algorithm is inspired by the work of Stein & Décoret [48], wherein a cost function is minimized. The cost function contains terms that quantize how good a position is for a given label.

To get a good result in the third dimension, the whole scene is divided into layers along the viewing direction. For each layer the optimal label placements are calculated. Temporal coherency is achieved by using forces to move the labels, instead of directly placing the label on the best position for the current frame.

This document is structured as follows: Related work concerning labeling and rendering of volumes and many transparent objects is summarized in Section 2. The algorithm is detailed in Section 3 as well as the used rendering techniques. Implementation details, primarily concerning optimizations and the rendering pipeline, are given in Section 4.

The resulting algorithm is used with real datasets and also evaluated with CAD drawings as well as arbitrary 3D scenes as presented in Section 5.

In Section 6 the results are interpreted and the overall work is discussed. Finally, Section 7 provides an outlook to possible future directions and improvements.

# 2 Related Work

## 2.1 Rendering

### 2.1.1 Volumes

Visualization of medical volume data can be divided into two categories: surface rendering and volume rendering. A good compendium of medical image visualization techniques, including both categories, is given by Preim and Botha [39].

For this work we implement direct volume rendering by means of ray casting. A survey of volume rendering techniques for GPUs is described Silva et al. [44]. We mostly follow the ideas outlined by Krüger and Westermann [21], like using a 3D texture to store volume data and rendering of front and back faces of the bounding volume to get the ray directions. The inspiration for using a texture atlas is taken from Kainz et al. [17]. This paper describes a whole volume rendering system implemented on the GPU using CUDA.

### 2.1.2 Transparency

Different approaches to render many transparent objects are compared by Maule et al. [31]. One way to implement fragment sorting is to store a linked list for each fragment in a so-called *A-buffer* (accumulation buffer). This requires more storage and a second pass to sort the linked lists. Lefebvre et al. [22, 23] propose an approach which improves upon the A-buffer, by replacing it with a *HA-Buffer* (hashing accumulation buffer). The HA-Buffer does not store more elements per fragment, but represents storage for all fragments regardless of screen position. The storage is filled by using a coherent hash function to determine the insertion place for a fragment. The fragments for a screen position are sorted while inserting them into the HA-Buffer. The sorting is an intrinsic property of the hashing function and stems from encoding the depth directly after an age counter (number of collisions) and using this combination as input integer of the hashing function.

## 2.2 Labeling

To provide more context to renderings, a couple of approaches have been described. For example, a context-preserving hotspot visualization technique by Krüger [20] renders the region around the mouse pointer with more transparency to peek into the object, while preserving the region around it for context. Another method is to explode the view to reveal the region of interest, as described by Bruckner and Göller [6]. A thorough treatment of interactive illustrations and animations to describe complex spatial relations is given by Preim [38]. We chose to annotate interesting locations in the scene and add labels with textual descriptions. The annotation points are called *anchors* and are represented by spheres in our system. The textual representation is called *label* and is associated with a line, called *connector*, to the anchor.

### 2.2.1 Origins

Labeling is a topic which has been researched for decades. It was first studied primarily in the context of cartography. Yoeli [56] and Imhof [15] have written early, influential publications. Imhof describes general principles and requirements for positioning labels on maps. For example, the object and label association should be easy to recognize. Labels should occlude as little other map information as possible. These aspects also apply to our labeling context.

The main challenge is that the problem of finding the optimal position for labels is NP-complete, as proven by Marks and Shieber [30]. This can intuitively be explained due to the fact that an algorithm must evaluate all possible positions for all possible label orders, because changing the order changes the result. The real proof is performed by reduction of the 3-SAT problem (see Gary and Johnson [10]) to label placement.

### 2.2.2 Internal and External Labeling

Generally, labels on maps are placed directly onto the object which is labeled. This is called internal labeling. Ropinski et al. [42] describe the advantages and also limitations of internal labels. It is hard to find an appropriate position for transparent objects and, in particular, for volumetric objects, because they usually have no defined surfaces. Also we do not want to occlude the point of interest. External labels are labels placed outside the object, and a line is drawn to the anchor for better visual association. A survey of labeling methods used in medical visualizations is given by Oeltze-Jafra and Preim [35]. These include algorithms for 3D objects as well as 2D images (specifically 2D slices).

### 2.2.3 Forces

One common method to find good label positions is to formulate criteria using forces and applying the forces to the labels so that they converge to a good position. This approach is introduced by Hartmann et al. [12]. They derive the forces from repulsive and attractive potential fields. Forces are used as a position optimization step by Stadler et al. [47] for geographic information systems. Pick et al. [36] use forces for virtual immersive environments, which are based on occlusions between objects. Temporal coherency can be improved by introducing a friction force, as demonstrated by Vaaraniemi et al. [53]. Their implementation is parallelized and utilizes the GPU.

### 2.2.4 Alternatives

Other fields in which label placement is researched, are virtual and augmented reality. Bell et al. [4] use the upright rectangular extents of objects projected into the view plane as representation and prevent occlusions in this domain by changing position, size and transparency. Additional information to determine better label positions can be produced by using a visual saliency algorithm, as described by Grasset et al. [11]. This paper also shows adaptive depth cues to further enhance the result.

A recent approach for external label placement is introduced by Tatzgern et al. [50], where each connector line is treated as pole and a label is restricted to movement along and rotation around this pole. This leads to more coherent results, but does not work for internal labeling when the camera is inside the object of interest.

### 2.2.5 Cost Function Minimization

Christensen et al. [8] provide a good overview of approaches which use an objective function for finding the appropriate label positions. The problem of local minima is investigated, and two approaches, gradient descent and simulated annealing, are proposed. Preuß et al. [40] formulate the principles of Imhof in terms of a fitness function. They use an evolutionary algorithm to determine the label positions with the fitness function as a heuristic to determine good positions and stopping criterion.

Stein and Decorèt [48] also use an energy function as formulation of the problem, albeit for 3D scenes. Their main contribution is a solution to the problem by using a sophisticated greedy algorithm. They propose an insertion order for the labels, which starts with the innermost (most surrounded) labels and gradually moves outward. It is based on the premise that the most surrounded labels are most difficult to place. The order is computed by leveraging an Apollonius diagram, also known as additively weighted Voronoi diagram [18]. The algorithm yields good results for opaque objects, but does not account for transparency or volumetric data. A drawback is the requirement for user-specified data for each surface to specify its importance. Another example of a greedy algorithm used for medical applications, albeit for 2D slices, is detailed by Mogalle et al. [33], wherein different sorting criteria are implemented and evaluated.

This work has been created as an evolution of a forensic case analysis tool, called `VolyRenderer`, developed at the Ludwig-Boltzmann-Institute for Clinical-Forensic Imaging at Graz University of Medicine by Urschler et al. [52].

# 3 Methods

## 3.1 Overview

Since most medical data is produced by means of MRI and CT, the resulting volumetric dataset is visualized using ray casting (ray tracing with secondary rays would be too expensive). Due to the size of the discretized 3D space, an exhaustive search on every possible 3D position for each label, is unfeasible. If the depth according to the volume or screen size is, for example, discretized into 1024 possible values, the time to minimize one label position would be increased by a factor of 1024.

Our main idea is to slice the 3D space into layers, parallel to the camera's near image plane. For each layer, the volumetric data is rendered up to the plane separating it from the next layer. Using this intermediate result, the labels with anchors in the layer are placed using an exhaustive search in 2D space. This exhaustive search yields optimal label positions and is described in the next section. The operation to divide the scene into layers is the subject of Section 3.3.

To prevent labels from jumping between their optimal positions in subsequent frames, the placement result is not used directly, but utilized as input for a force framework. These forces update the positions of the labels, as detailed in Section 3.4.

Finally, in Section 3.5, our approach to rendering the whole scene, including volumes and transparent objects, is portrayed.

A graphical overview of the main stages of the program is shown in Figure 3.1. All major steps are represented as blocks. For further details about the implementation consult Section 4.

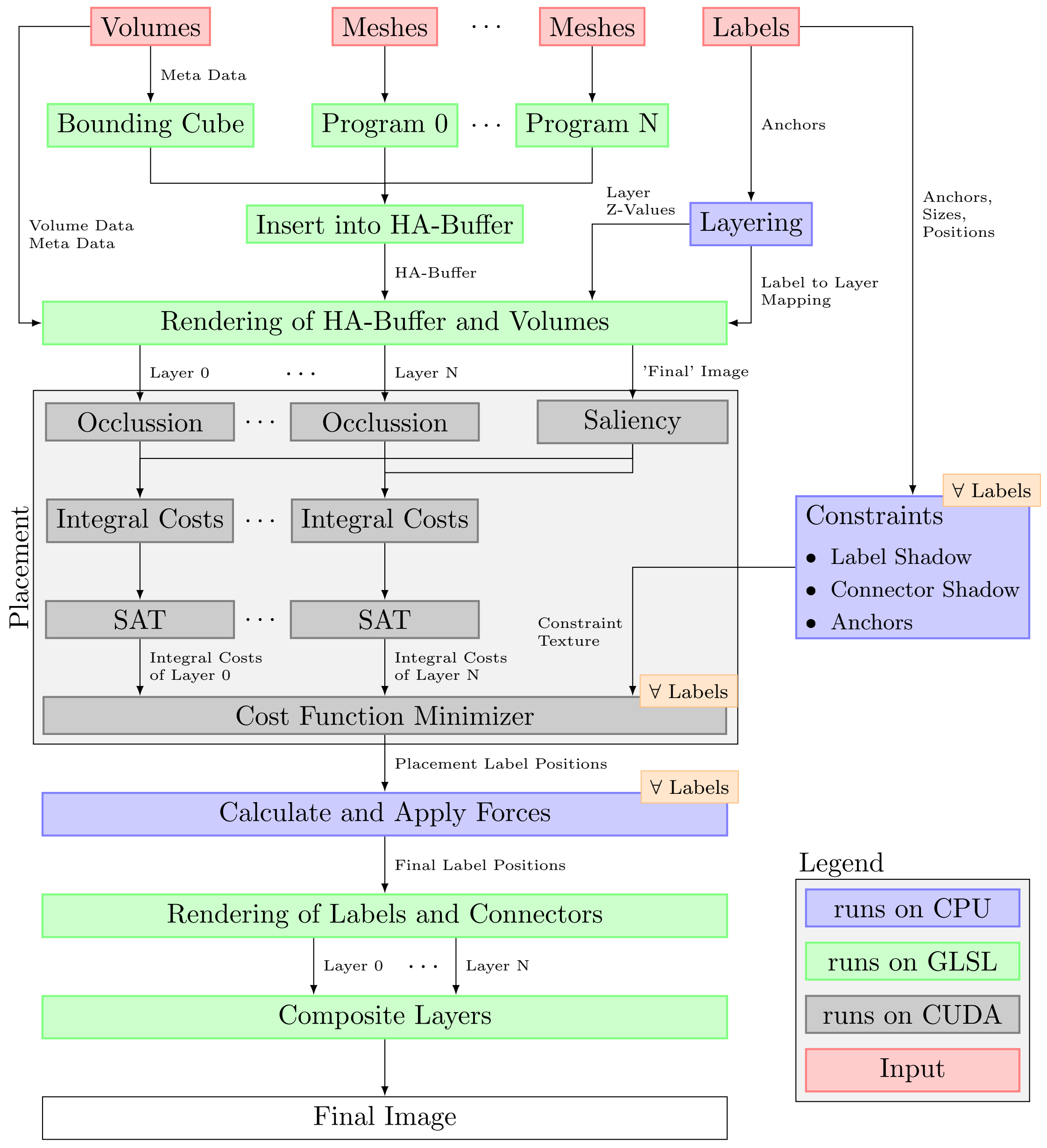


**Figure 3.1:** Schematic overview of the algorithm. It shows the rendering stage of volumes and meshes at the top, which is implemented using an HA-Buffer. The results are N rendered layers and a “final” image without labels. Next, the best label positions are calculated using cost minimization. Finally the labels and connectors are rendered and the final image is generated. The processor, GPU or CPU, and programming environment, CUDA or OpenGL Shading Language (GLSL), are color-encoded.

## 3.2 Placement

Label placement is the task to find the best suitable position for each label. How “good” a label position is, is determined by certain criteria. These criteria are formalized mathematically, so that for each possible label position (each coordinate in the screen space or search space) a cost or energy value can be calculated. Thus, finding the best label position is an energy minimization problem. This basic idea is described by Stein and Dècoret in [48]. We adopt their energy for a label $i$ as follows:

$$\begin{aligned} E_i(x_i, y_i) = \alpha &\left| \boldsymbol{a}_i - \hat{\boldsymbol{l}}_i^{(j)} \right| \\ +\ &\beta \left( |\cos\theta_i| + |\sin\theta_i| \right) \\ +\ &\gamma\, f_{label}(x_i, y_i) \\ +\ &\delta\, f_{connector}(x_i, y_i) \\ +\ &\epsilon\, f_{anchor}(x_i, y_i) \\ +\ &\zeta \iint_{B(\boldsymbol{l}_i, \boldsymbol{s}_i)} g(\mu, \nu)\, \mathrm{d}\mu\, \mathrm{d}\nu \\ +\ &\eta\, \| \hat{\boldsymbol{l}}_i^{(j)} - \boldsymbol{l}_i^{(j-1)} \| \end{aligned} \tag{3.1}$$

The first term in (3.1) is the distance between the anchor position $\boldsymbol{a}$ and the currently evaluated position candidate $\hat{\boldsymbol{l}}_i^{(j)} = \begin{bmatrix} x & y \end{bmatrix}^{\mathrm{T}}$ for label $i$. Putting the label as close as possible to the anchor leads to a better visual association. This is one of the principal guidelines for cartography, as introduced by Imhof [15]. $j$ is the frame index and denotes the current frame.

The second term favors horizontal and vertical connector lines. The angle of the connector line to the horizontal axis is given by $\theta$.

In the following three terms (with the weights $\gamma$, $\delta$ and $\epsilon$) different constraints, to prevent overlapping of labels, connectors and anchors, are included. Those are detailed next, in Section 3.2.1.

The second to last term is used to get a value for costs that are depending on the area of the whole label. This can be described as an integral of the function $g(\mu, \nu)$ over the label box. The box is defined by the label position $\boldsymbol{l}$ and the label's size $\boldsymbol{s}$, which consists of its width and height. The integral costs are described in Section 3.2.2.

Finally, the last term is the distance between the currently evaluated position and the final position of the last frame $\boldsymbol{l}_i^{(j-1)}$. The goal is that the label does not try to move to a very distant target position, but has a smooth motion of its own.

Each term has an associated weight ($\alpha$–$\eta$), to tune the behavior of the system. All building blocks of the placement algorithm are shown in the “Placement” box in Figure 3.1.

The goal is to minimize the sum of the energies for all labels, as stated in (3.2), but this is not feasible for real time applications. Instead each label energy is minimized on its own.

$$\min_{x_i, y_i} \sum_i E_i(x_i, y_i) \tag{3.2}$$

### 3.2.1 Constraints

Three constraints are formulated so that a newly placed label does not occlude elements of other labels. These elements are:

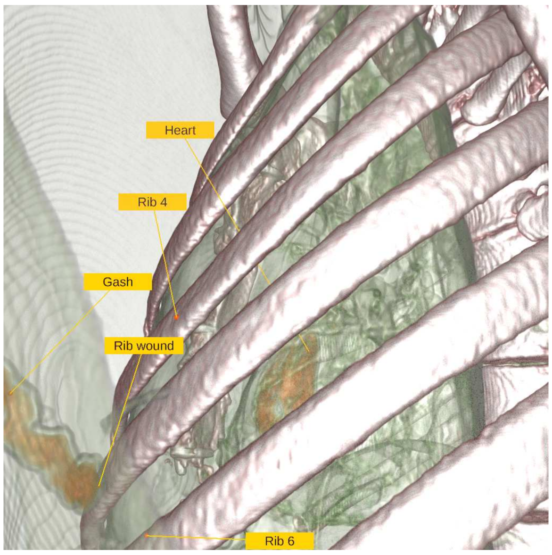
- label boxes,
- connectors and
- anchors.

To prevent the new label from occluding an already positioned label and to prevent the connector of the new label from crossing other labels, the following steps are taken. The new anchor position is used as a virtual light source and the shadow of each already placed label is calculated. This shadow region is dilated with a box of the size of the new label plus an additional margin (Figure 3.2d). The same procedure is applied to the shadow of already existing connector lines, see Figure 3.2c. These two constraints are taken from Stein and Décoret [48].

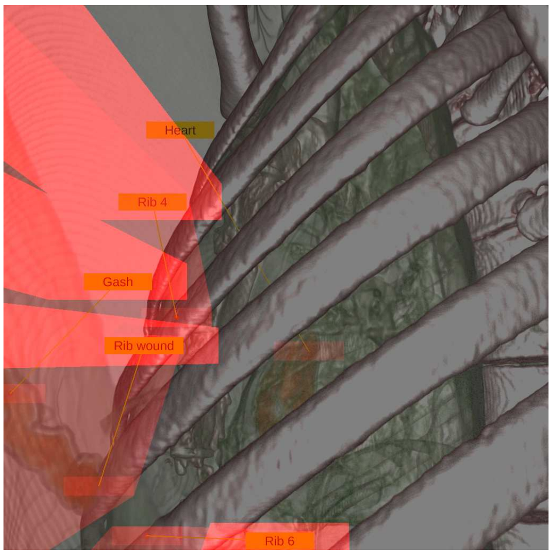
Additionally a rectangle with the size of the new label plus an additional margin is drawn on each anchor position irrespective of whether the label was already placed. All these steps and their results are displayed in Figure 3.2. In the overlay image (Figure 3.2b) it might seem that the “Heart” label violates the constraints of “Rib 4”, but the label’s center — which is the only point evaluated for a label — is outside the constraint.

The constraints calculation is placed on the right side in Figure 3.1. It only depends on label data (anchor, size and position) as well as margin sizes.

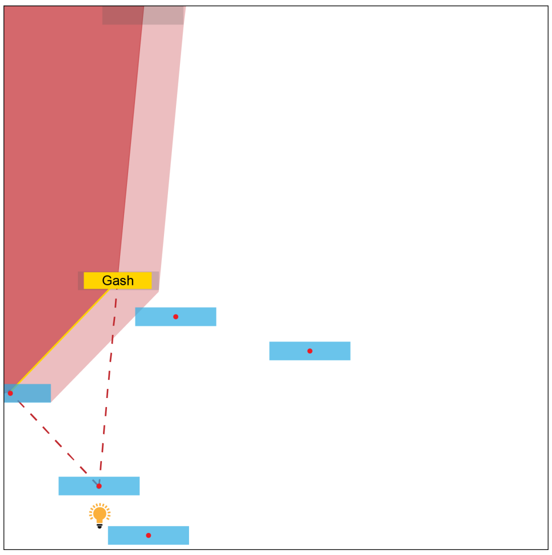
In contrast to Stein and Dècoret the constraints are not “hard”, in the sense that no constraint violation is possible, but “soft”. This has the advantage that there exists always a “best” position for each label at the cost that this position might, in fact, be rather bad. The other approach would be to hide labels which violate the constraints, or put them at a certain location (as Stein and Dècoret do). Another advantage of this formulation is, that the constraints can be weighted differently (also depending on the system state), or can even be disabled for certain situations.



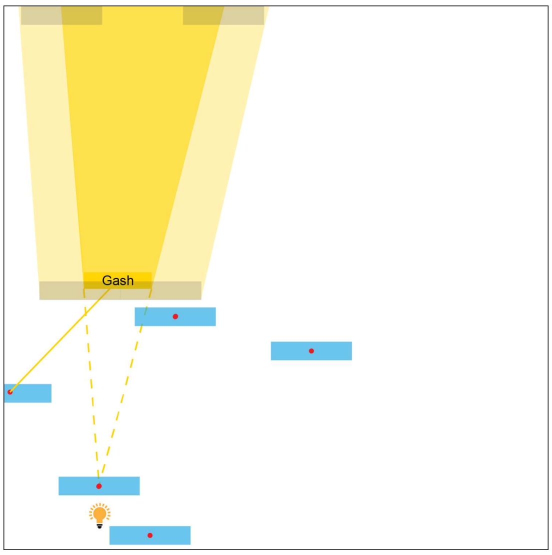
(a)



(b)



(c)



(d)

**Figure 3.2:** Example image with visualizations of constraints. (a) is the final image. (c) and (d) are constraint illustrations for the second label "Rib wound", after the first label "Gash" has already been placed. Anchors are denoted by red dots. The blue boxes are the anchor constraints. (c) is a conceptual drawing of the connector constraint, (d) of the label constraint. The darker areas show the shadow of the connector and label. The lighter margin encloses the final constraint region and visualizes the dilation with the new label. In (b) the constraints for the "Heart" label are shown as overlay on the scene. The label box shadow constraint is displayed in bright red and the connector shadow in red. Illustrations kindly created by Martin Hatzl.

### 3.2.2 Integral Costs

The integral costs are given by combining occlusion and saliency as follows: (also see "Integral Costs" in Figure 3.1):

$$g(x,y) = \sum_{i=0}^{l} o(x,y)_i + \big[\beta \cdot s(x,y) + (1-\beta)\big] \cdot \sum_{i=l+1}^{n-1} o(x,y)_i \tag{3.3}$$

Where $l$ is the current layer index, $n$ is the number of layers, $o(x,y)_i$ is the occlusion of layer $i$ and $s(x,y)$ is the saliency. The integral cost is the sum of the occlusions of all layers up to the current one plus the weighted sum of the occlusions behind the current layer. The weighting depends on the saliency and on the parameter $\beta$, which defines how much the saliency influences the result. An example is given in Figure 3.3. The background, which can be seen through the skin in the upper left corner, has the lowest weight. Edges around the ribs have a high value, as well as the detail of the heart behind the ribs.

When only one layer is used, a different function is used to combine occlusion and saliency, because (3.3) would collapse to only the occlusion term:

$$g(x,y) = \alpha o(x,y) + \beta s(x,y) \tag{3.4}$$

In (3.4) the occlusion and saliency measures are weighted by $\alpha$ and $\beta$ respectively and added together.

The saliency $s(x,y)$ describes how much a pixel stands out compared to its neighbors. This is a useful metric to describe how much information is present in an image region. Labels should be placed in such a way that as little information as possible is covered by them. A crude saliency estimation can be realized using the magnitude of the image gradient:

$$s(x,y) = \|\nabla i(x,y)\| \tag{3.5}$$

The gradient in turn is approximated using the Sobel kernel [46]. Since the source image in the application is in the RGBA color space, and the Sobel kernel is only defined for a matrix of scalars, we cannot apply it directly. First the RGBA image is converted to CIE XYZ and then to CIELAB [43, 51]. CIELAB is used because it is perceptually uniform [16, p. 73]. For each of the three components of CIELAB, the horizontal and vertical derivative approximations are calculated. These are then represented as two three-dimensional vectors. Finally, the Euclidean distance of the two vectors is used as the result. A better result could be achieved by using the Canny edge detector [7] (used by Grasset et al. [11]), but at a considerable performance cost.

The occlusion $o(x,y)$ describes how much information a pixel obscures. As a cheap approximation, the alpha value of the RGBA color at each pixel is used. This describes the opacity of the color and hence indicates how much of the underlying layers it occludes. For a layer, the alpha value of the layer plus the alpha values of the layers behind it are used.

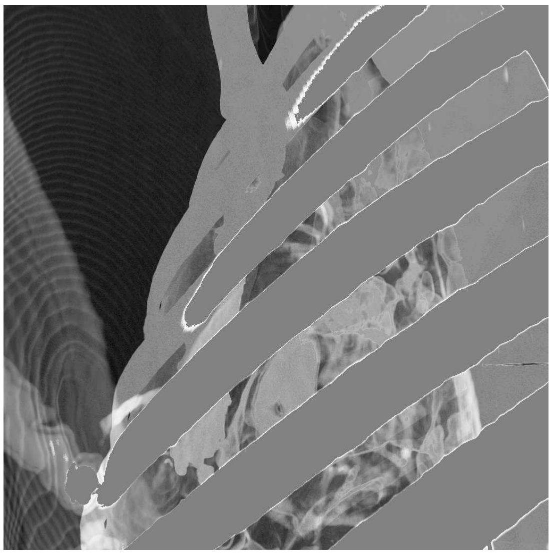


**Figure 3.3:** Integral costs image for the second layer of Figure 3.2a.

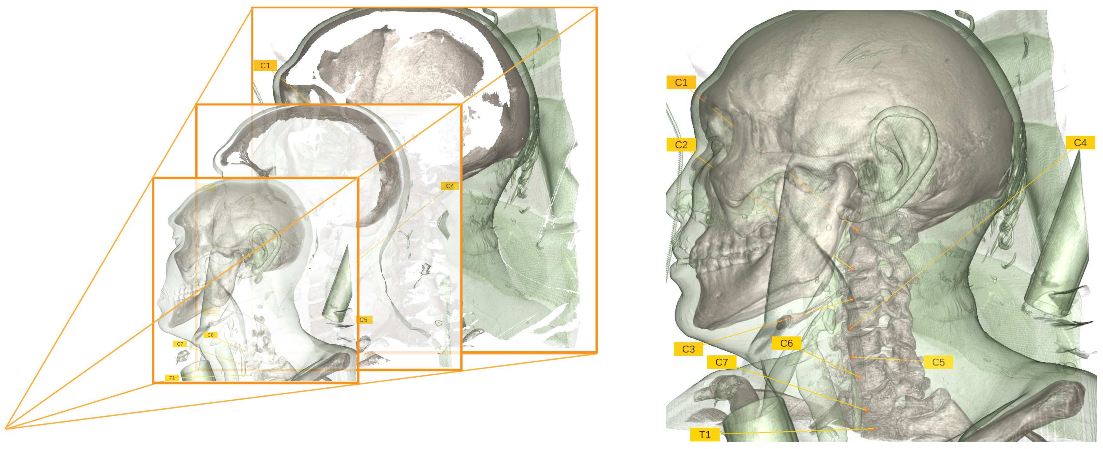
### 3.2.3 Algorithm

Now that we have a way to get the "best" position for a label $i$ using (3.1), we are able to calculate the positions of all labels in a layer.

In the default mode, the label iteration order is the insertion order. For each label the constraints are updated (see Section 3.2.1), the cost function is minimized and the label's position is determined. This is a greedy approach, which means that the overall result is highly dependent on the order of the labels.

The second mode is a reimplementation of Stein and Dècoret. They propose the usage of a greedy algorithm to achieve real-time performance. It uses a more sophisticated approach to process the innermost labels first. This is achieved by using the Apollonius diagram, which is also known as additively weighted Voronoi diagram, of the anchors (see [48]). The rationale behind this is the assumption that labels that are surrounded by many anchors and far away from the background, are harder to place. This is probably the case for most scenes with outside placement on the background, but not for inside placement, when there is no visible background left. The algorithm of Jump Flooding [41] is used to create the distance transform as weight for the Apollonius diagram and the diagram itself. Applying this order also leads to more label movement, due to more frequent changes of each optimal label position. This is later discussed in Section 6.

In a third mode the label order is chosen randomly for each layer and the energies for all labels are summed up. The resulting value is compared to the sum of the last label ordering. If the new order has an overall lower energy, it is used, otherwise it is discarded. This yields very good results, but is only usable for a static camera position. It is too slow



(a) (b)

**Figure 3.4:** Illustration of the result image of each layer in the context of the whole 3D scene in (a). Kindly created by Martin Hatzl. (b) shows the overall result image.

for using it in an interactive environment, see the measured performance in Table 5.2. A good use case for this mode is to create good images of certain key perspectives of the scene. This mode essentially brings us one step closer to fulfilling the minimization in (3.2), but only for each layer separately.

## 3.3 Layering

In contrast to polygonal scenes or augmented reality (AR), volumetric scenes pose more complexity for label placement through partially transparent objects. This is a challenge, but also yields some opportunities:

- more space for label placement, if labels are allowed to be partially occluded
- labels can be placed closer to the anchor
- better placement due to additional depth information
- additional depth cues: different label colors, connector can partially be occluded

It is infeasible to create one layer for each anchor. In order to approximate the depth as good as possible for the label placement, it is important that the layers are chosen wisely. The layer of each label should be as close to the anchor depth as possible, so that the information derived from the layer result is as accurate as possible. We cluster the anchor points using their depth values. This is done after projecting the anchor points into the non-linear z-space to gain more precision near the virtual camera. Figure 3.4 shows how the overall result is built up from the layers.

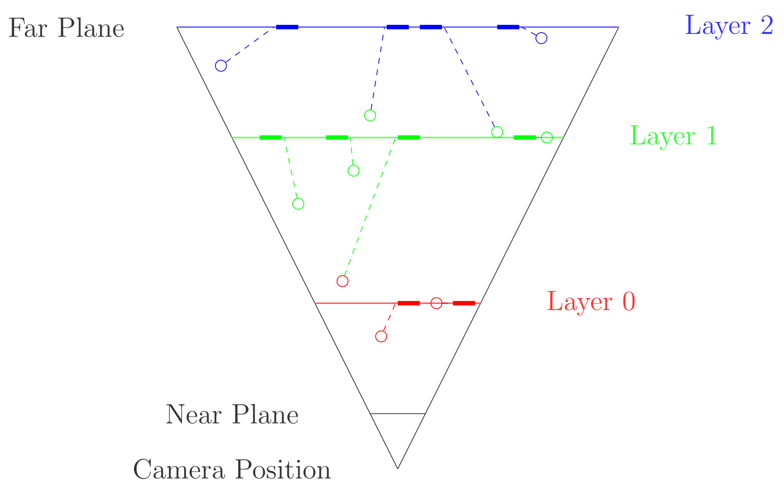


**Figure 3.5:** Conceptual visualization of clustering, derivation of layers and placing labels on the layers. The illustration shows a top-down view of a scene. Anchors are represented as circles. Three clusters are formed and marked by the colors of the anchors (red, green and blue). The layers are placed at the mean depth of the anchors. Labels are represented by horizontal lines, colored with the layer color. The connectors are shown as dashed lines from the anchor to the label.

For clustering the, `k-means++` algorithm is used, as presented by Arthur and Vassilvitskii[3]. This algorithm improves upon the original `k-means` algorithm [26, 28, 49] by changing the seeding of the clusters. Instead of choosing the seeds at random, only the first cluster center is chosen at random using a uniform probability distribution. Each subsequent seed is chosen of the remaining anchors with a probability weighted by the squared distance to the nearest already chosen cluster center. This causes the cluster centers to have a larger distance between each other. Overall, the clustering results improve considerably. With the standard `k-means` algorithm, often not all available clusters were used, which was caused by merging of clusters during the loop that reassigns anchors to the nearest cluster center.

After clustering the anchor points, the layers are placed at the median z-value of all anchors in each cluster, so that there is one layer for each cluster (see “Layer Z-Values” edge from “Layering” box in Figure 3.1). This prevents outliers from influencing the layer position and guarantees that at least for one label all depth-dependent terms are exact and not only approximations. A label is associated to a layer, if it is on the layer or between the layer and the next layer toward the camera position. Each label’s position is altered so that it is on the associated layer by changing only the z-value (see Figure 3.5). The label could now be drawn onto the layer without the need to perform ray-tracing, because of the intermediate layer results. In our case, the labels are not yet drawn, but the association from each label to a layer is passed to the rendering part as shown in Figure 3.1. This is because a label’s final position is determined by forces.

## 3.4 Forces

To provide temporal coherency, a stripped down forces framework is used, which is depicted in the context of the whole algorithm by the “Forces” box in Figure 3.1. It is inspired by Hartmann et al. [12], where attractive and repulsive forces act on each label depending on its position. The forces influencing a single label are added together, and the position is updated accordingly.

We just use one force, which is derived for each label by its ideal position determined by the placement algorithm.

$$\boldsymbol{d}_i = \boldsymbol{p}_i - \boldsymbol{l}_i \tag{3.6}$$

In (3.6), $\boldsymbol{d}_i$ denotes the vector from the current label position to the placement position. It is used to calculate the force as follows:

$$\boldsymbol{f}_i(\boldsymbol{l}_i, \boldsymbol{p}_i) = \begin{cases} \dfrac{\boldsymbol{d}_i}{\|\boldsymbol{d}_i\|} & \text{if } \|\boldsymbol{d}_i\| \geq \epsilon \\ \dfrac{\boldsymbol{d}_i}{\epsilon} & \text{else} \end{cases} \tag{3.7}$$

The force $\boldsymbol{f}_i$ in (3.7) is a function of the current label position $\boldsymbol{l}_i$ and the placement result $\boldsymbol{p}_i$. If the distance to the target position is smaller than $\epsilon$, the force is not normalized anymore, but attenuated more and more towards the target position. This yields a more natural motion by interpolating the force’s magnitude from 1 to 0 towards the placement position. It also prevents overshooting the destination.

In each frame, the force is applied by multiplying it with a time value $t$ and adding the result to the current position $\boldsymbol{l}_i^{(j)}$ to yield the new label position $\boldsymbol{l}_i^{j+1}$:

$$\boldsymbol{l}_i^{(j+1)} = \boldsymbol{l}_i^{(j)} + t\,\boldsymbol{f}_i \tag{3.8}$$

The time span $t$ is the computation duration of the last frame. This causes smooth transitions of the labels, if the duration is short enough. If it is higher, the labels tend to jump around the desired destination positions, given by the placement result. To circumvent this problem, the forces can also be simulated multiple times per frame using a lower time value and applying the results multiple times. This is, in essence, a numeric approximation of an integral of the force over the time.

Another trick to prevent unpleasing label jumping around the target position, is to artificially move the label to the placement result position, if it is already in a certain neighborhood of the target.

## 3.5 Rendering

The main requirements for the rendering system are to render complex scenes involving transparency and to be able to display multiple volumes. Both requirements are detailed in the following sections respectively.

### 3.5.1 Transparent Objects

Rendering of transparent objects behind each other requires sorting of all the fragments at the pixel in screen coordinates from front to back. When traversing fragments $\boldsymbol{p}^i$ — given as vectors consisting of red $p_r^i$, green $p_g^i$, blue $p_b^i$ and alpha $p_a^i$ components — in this order, the color after each step $\boldsymbol{c}^{i+1}$ is given as:

$$\boldsymbol{c}^{i+1} = \boldsymbol{c}^i + (1 - c_a^i) \cdot p_a^i \cdot \begin{pmatrix} p_r^i \\ p_g^i \\ p_b^i \\ 1 \end{pmatrix} \tag{3.9}$$

This is called blending equation for in-order traversal, or over-operator [37]. Alternatively, one could sort them from back to front and adjust (3.9) accordingly. This would have the disadvantage of not being able to implement the early out optimization (see Section 4.2.2).

### 3.5.2 Volumes

There are many techniques to render volumes using the GPU (see Silva et al. [44] and Kainz et al. [17]). We use a single 3D texture to store all volume data. A border is added around each single datasets in this volume atlas to allow algorithms (for example calculating the gradient) to access voxels just outside the volume, so that no additional border handling has to be implemented. Multi volume ray casting is implemented, as illustrated in Section 4.2.2.

# 4 Implementation

In this chapter significant implementation details are revealed. Section 4.1 is about the placement implementation and section 4.2 about rendering, divided into transparent objects and volume rendering. Finally, section 4.3 lists the memory requirements of the implemented algorithms.

## 4.1 Placement

### 4.1.1 Constraints

All three constraints — for labels, connectors and anchors — are rendered into a single byte buffer. The constraints are bit-encoded, which is achieved by using OpenGL's logic operators. Dilation, which is the same as a *Minkowski Sum*, with an arbitrary structuring element is a very costly operation [9, 55]. Therefore, an optimized geometry shader for dilation with a rectangle is used. It is inspired by a conservative rasterization algorithm for vertex shaders [13].

The geometry shader receives two edges ($e_1$ and $e_2$) of the shadow polygon at once. It has the task to dilate along the edges with the new label as structuring element. Depending on the edge configuration, three different cases must be handled, which lead to a different number of emitted vertices. These are shown in Figure 4.1.

If the normals of both edges $n_1$ and $n_2$ are in the same quadrant, as in Figure 4.1a, the dilation region can simply be created by emitting a new vertex, which has an offset of half the width and height of the new label (half-diagonal).

When the normals are in adjacent quadrants, two edge vertices must be created. Two different signs for adding the half-diagonal elements are used, depending on which two quadrants are occupied as shown in Figure 4.1b.

Finally, if the normals are in opposing quadrants (Figure 4.1c), three edge vertices have to be used. The signs are again determined by the quadrants. The signs for the third vertex can also be calculated from the normals.

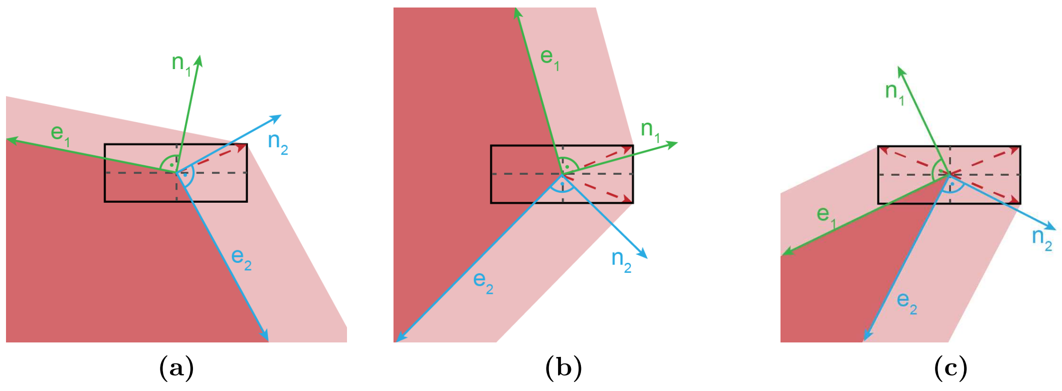


**Figure 4.1:** Three different cases of the dilation process. (a) requires no new corner vertex, (b) demands one additional corner vertex and (c) calls for two more corner vertices. Illustrations kindly created by Martin Hatzl and inspired by graphics in [13].

This increased the performance by a factor of three, compared to an optimized implementation on the CPU using Clipper `http://www.angusj.com/delphi/clipper.php`. The new timings are shown in Section 5.4 in Table 5.2 in the "placement column".

### 4.1.2 Integral Costs

Both integral cost components (saliency and occlusion) are implemented using CUDA. `cudaTextureObject_t` and `cudaSurfaceObject_t` objects are used to access read-only or read- and writable memory, respectively. This provides a unified interface and hides the underlying memory type, which could either be a OpenGL texture or some sort of `cudaArray`. Using a texture provides the required integration with OpenGL, but also has faster access times in a pixel's local neighborhood.

Calculating the sum of the integral cost pixels during minimization would be quite inefficient, because for each screen coordinate (possible label position), the area in the label box would have to be sampled. Using a *summed area table* (SAT), these redundant calculations can be removed [34]. For each pixel with coordinates $(x, y)$, the SAT stores the sum of the rectangle defined by the points $(0, 0)$, $(x, 0)$, $(0, y)$ and $(x, y)$. Using this SAT the sum for all pixels in a rectangle can be calculated by only four texture lookups for the corner pixels. The sum is then given by adding the upper left and lower right corner values and subtracting the other two corners. The creation is implemented in CUDA and follows the algorithm introduced by Nehab et al. [34].

## 4.2 Rendering

The view frustum is optimized to only fit the objects of the scene. This increases the depth precision during rendering and helps to prevent z-fighting. See Vasilakis and Papaioannou [54] for an explanation of z-fighting and alternative ways to circumvent it.

### 4.2.1 Transparent Objects

We adhere to the basic HA-Buffer implementation of Lefebvre et al. [23] and extend it to support intermediate layer results and volume rendering. We store an object identifier and the position in eye space in addition to the color for each fragment. The position is used to determine onto which layer the fragment must be drawn and to construct rays for volumes. The identifier is used to distinguish between different volumes. Since the HA-Buffer is also well suited for *constructive solid geometry* (CSG), the identifier may also be used to aid in realizing this feature.

### 4.2.2 Volumes

The first step during volume rendering is to render the bounding volume, in order to get ray directions and texture coordinates, as described by Krüger and Westermann [21]. But instead of rendering the front and back face in two passes, we use the capabilities of the HA-Buffer. Thus, only one pass is needed. The ray directions and texture coordinates can be retrieved from the positions in the fragment data by applying transformation matrices. To increase the performance by preventing unnecessary computations, an intersection between the near plane and the bounding volume is calculated using a geometry shader. Now, the rays start at the near plane, and no visibility test has to be performed.

The volume rendering is integrated in the shader which traverses the fragments from the HA-Buffer. If a fragment from a volume is found during traversal, it is stored. A ray from this stored fragment to the next fragment of the same screen position is constructed. The second fragment might be the back face of the bounding volume or a fragment from an intersecting object. In any case, points along the ray are sampled from the 3D texture. The intensity of the texture is converted into a color representation using a transfer function. For lighting, the gradient of each point is calculated using central differences, and Blinn-Phong shading [5] is performed [14, 24].

For multi-volume rendering, all active objects are tracked. These are objects where the front face fragment has already been processed, but the back face fragment has yet to be encountered. For each active volume, the sample color is determined as outlined above. All those colors from different volumes at the same sample position are then accumulated. Samples along a ray are composited using the blending equation (3.9).

If the alpha value is greater than a certain threshold during sampling along the ray, the sampling is stopped, and the current color value is used as final result. This is called early ray termination and was introduced by Levoy [25], see Listing 1 line 28.

The following two Listings 1 and 2 show a shortened version of the volume rendering code.

```
vec4 calculateColorOfVolumes(in int activeObjects, in int activeObjectCount,
    in vec4 segmentStartPos_eye, in vec4 endPos_eye, in vec4 fragmentColor)
{
  float segmentTextureLength = calculateSegmentTextureLength(
      activeObjectCount, activeObjects, segmentStartPos_eye, endPos_eye);
  float sampleSteps = segmentTextureLength * STEP_FACTOR;
  sampleSteps = clamp(sampleSteps, 1.0f, MAX_SAMPLES - 1.0f);

  vec4 step_eye = (endPos_eye - segmentStartPos_eye) / sampleSteps;

  vec4 currentPos_eye = segmentStartPos_eye;

  // sample ray segment
  for (int stepIndex = 0; stepIndex < sampleSteps; ++stepIndex)
  {
    vec4 sampleColor = calculateSampleColor(activeObjects,
        activeObjectCount, currentPos_eye);

    // sample accumulation
    fragmentColor = fragmentColor + sampleColor * (1.0f - fragmentColor.a);

    if (fragmentColor.a > alphaThresholdForDepth)
    {
      setDepthFor(currentPos_eye);
    }

    // early ray termination
    if (fragmentColor.a > EARLY_RAY_TERMINATION_ALPHA)
      break;

    currentPos_eye += step_eye;
  }  // sampling steps

  return fragmentColor;
}
```

**Listing 1:** Ray casting

```
vec4 calculateSampleColor(in uint remainingActiveObjects,
    in int activeObjectCount, in vec4 currentPos_eye)
{
  vec4 sampleColor = vec4(0.0f, 0.0f, 0.0f, 0.0f);

  // sampling per non-isosurface object
  for (int objectIndex = 0; objectIndex < activeObjectCount;
      objectIndex++)
  {
    int currentObjectId =
      calculateNextObjectId(remainingActiveObjects);

    vec3 textureSamplePos =
      (volumes[currentObjectId].textureMatrix * currentPos_eye).xyz;

    float density = getVolumeSampleDensity(textureSamplePos);
    vec4 currentColor =
      transferFunctionLookUp(currentObjectId, density);
    vec3 gradient =
      getVolumeSampleGradient(currentObjectId, textureSamplePos);

    currentColor.rgb = clampColor(calculateLighting(
      currentColor, currentPos_eye.xyz, gradient));

    // we sum up overlapping contributions
    sampleColor += currentColor;
  }  // per active object loop

  return clampColor(sampleColor);
}
```

**Listing 2:** Sample color calculation

## 4.3 Memory

The memory requirements of the current implementation are listed in Table 4.1 for a view resolution of $1024 \times 1024$ pixels. The distance transform and Apollonius compute components are only required for the Stein &Décoret mode. The `struct` for the fragment data entails two vectors (`vec4`) for the color and the position as well as an `int` for the identifier. This could certainly be optimized depending on how many different volumes should be supported. The HA-Buffer size can be adapted depending on how many transparent pixels, that overlap each other in the depth direction, should be supported. For comparison, a volume for example with the size of $512 \times 512 \times 1024$ voxels takes up 1024 MB of memory. See Section 6 for more potential optimizations.

| **Component** | **Size** | **Format** | **Memory** | **Per layer** |
|---|---|---|---|---|
| | Pixel | - | MB | - |
| Layer render target | $1024 \times 1024$ | `RGBA16F` | 8 | yes |
| (Distance transform) | $1024 \times 1024$ | `R32F` | 4 | (yes) |
| (Apollonius compute) | $1024 \times 1024$ | `int` | 4 | (yes) |
| Accumulated layers | $1024 \times 1024$ | `RGBA16F` | 8 | no |
| Depth buffer | $1024 \times 1024$ | `DEPTH24_STENCIL8` | 4 | no |
| Integral costs | $1024 \times 1024$ | `R32F` | 4 | no |
| Summed area table | $1024 \times 1024$ | `float` | 4 | no |
| SAT temp | $65 \times 1024$ | `float` | 0.25 | no |
| Occlusion | $1024 \times 1024$ | `R32F` | 4 | no |
| Saliency | $1024 \times 1024$ | `R32F` | 4 | no |
| Constraint buffer | $1024 \times 1024$ | `byte` | 1 | no |
| **HA-Buffer:** | | | | |
| Records buffer | $5793 \times 7593$ | `long` | 256 | no |
| Counts buffer | $5793 \times 7593$ | `int` | 128 | no |
| Fragment data | $5793 \times 7593$ | `struct` of 36 bytes | 1152 | no |

**Table 4.1:** Memory usage breakdown

# 5 Results

Overall results for different scenes are presented in Section 5.1. The temporal behavior is displayed in Section 5.3. Section 5.2 compares our proposed algorithm with a Stein and Décoret reimplementation. Performance measurements and a stress test scene are shown in Section 5.4

## 5.1 Example Scenes

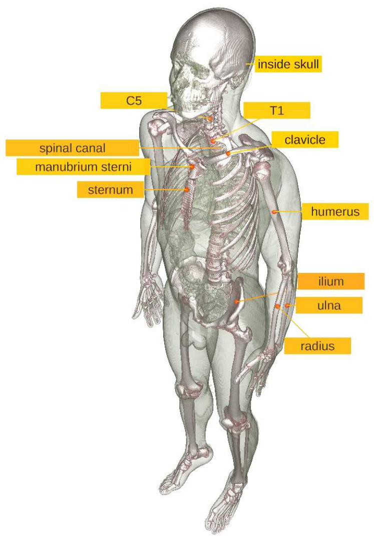
All results for volumetric scenes are obtained by using real world data. An overview of the used datasets and their sizes is given in Table 5.1. Some sizes may differ to the original datasets. This is due to optimizations and removal of artifacts, like metal artifacts or the CT table. These changes as well as the creation of the masks for the LIDC-IDRI lung scene (Section 5.1.5) and the knife scene (Section 5.1.3) were performed using the `Ambient Image Processor` tool [19]. The source code is available at `https://github.com/josefkoller/ambient_image_processor`.

### 5.1.1 Full Body CT-Scan

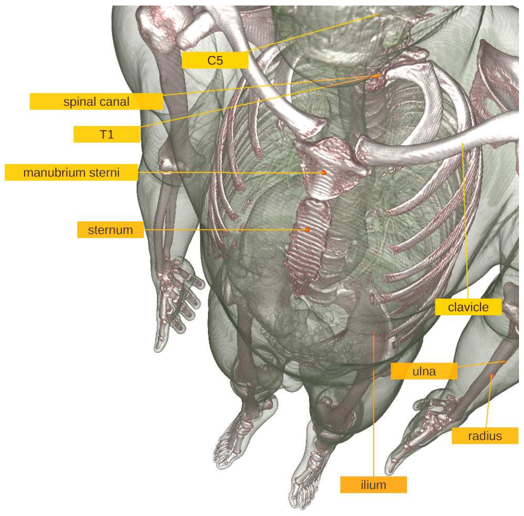
The first scene is a full body CT-Scan of a human taken from the Whole Body Morphometry Project [29]. Ten labels are placed at certain anatomically important points to create a test scene. The transfer function shows the bones and tissue boundaries (skin and inner organs). Pictures can be seen in Figure 5.1.

| **Scene Name** | **Width** | **Height** | **Depth** |
|---|---|---|---|
| | Pixel | Pixel | Pixel |
| Full Body CT-Scan | 160 | 340 | 1718 |
| Heidelberg Shot | 512 | 280 | 2001 |
| Heidelberg Knife | 400 | 256 | 1791 |
| - Air | 190 | 162 | 334 |
| Heidelberg Fall | 512 | 512 | 1051 |
| Heidelberg Motorcycle | 512 | 296 | 2107 |
| LIDC-IDRI Lung | 512 | 512 | 465 |
| - Lesion 1 | 28 | 32 | 24 |
| - Lesion 2 | 16 | 20 | 16 |
| - Lesion 3 | 20 | 18 | 16 |
| - Lesion 4 | 18 | 22 | 16 |
| Pedestrian | 512 | 512 | 2693 |
| Bus Victim | 512 | 222 | 447 |
| - Head | 512 | 400 | 746 |
| 100 Labels | 256 | 230 | 256 |

**Table 5.1:** Overview of volume datasets



**(a)**



**(b)**

**Figure 5.1:** Scene of a human, showing labels that are mainly placed on the background.

### 5.1.2 Heidelberg Shot

This scene is one of many "Heidelberg" scenes from the Institute of Legal and Traffic Medicine, University Hospital Heidelberg. It shows a suicide by a gunshot into the head. The gun is shown with a mesh, the expected trajectory and the real trajectory of the bullet are also visualized in Figure 5.2.

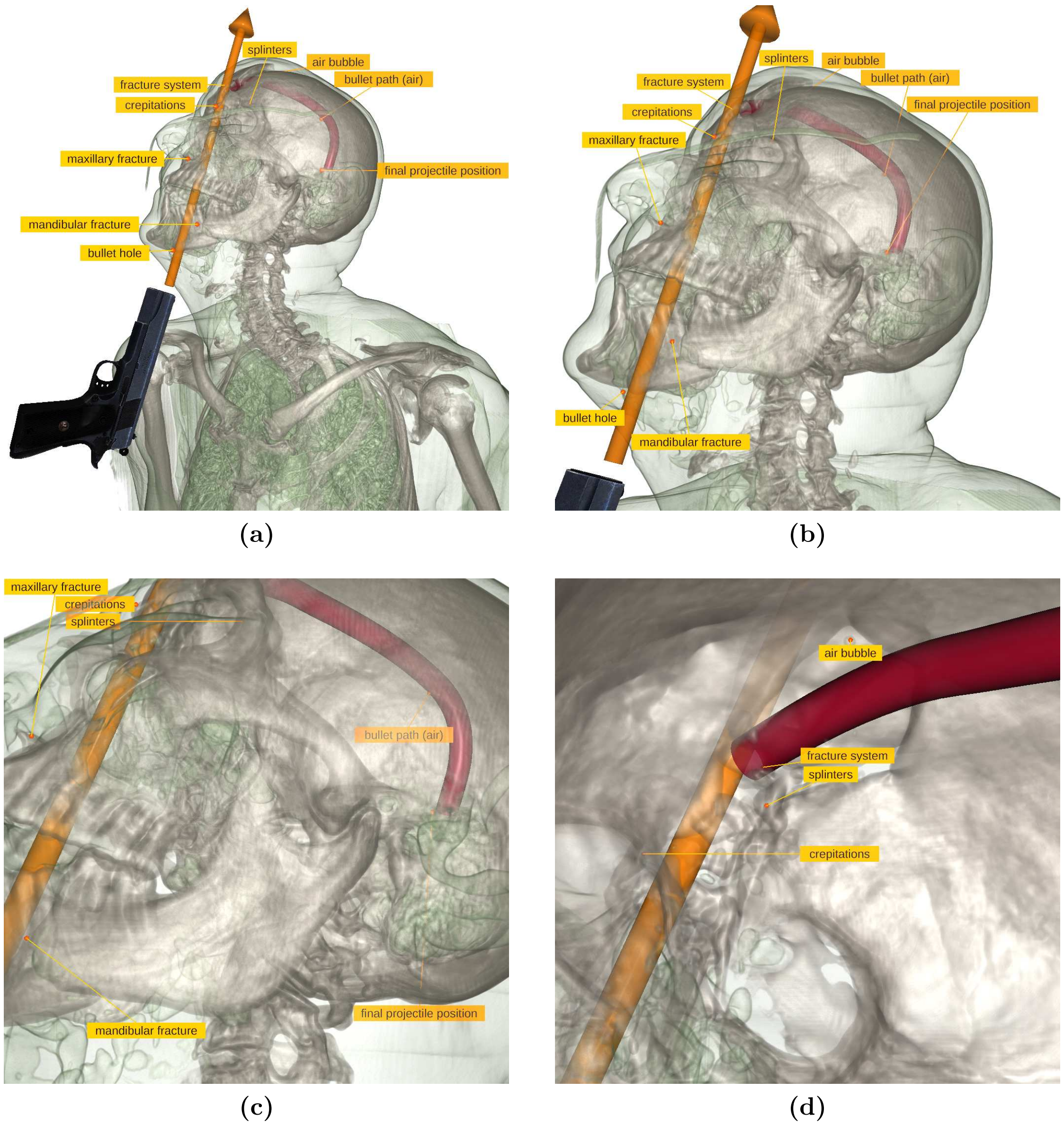


**Figure 5.2:** Images of a forensic scene. Meshes and labels are added to convey more information.

### 5.1.3 Heidelberg Knife

In this scene (Figure 5.3), a stabbing victim is shown. The air cavities (lung and wounds) are highlighted using segmented volume datasets for each cavity. The segmentation was achieved with the `Ambient Image Processor` tool by using region growing starting from seed points. The extra datasets have a different transfer function to highlight the air. This visualization can be used to reconstruct knife stabbings. The injuries and near anatomical features are annotated. The scene is also used in Figure 5.12c.

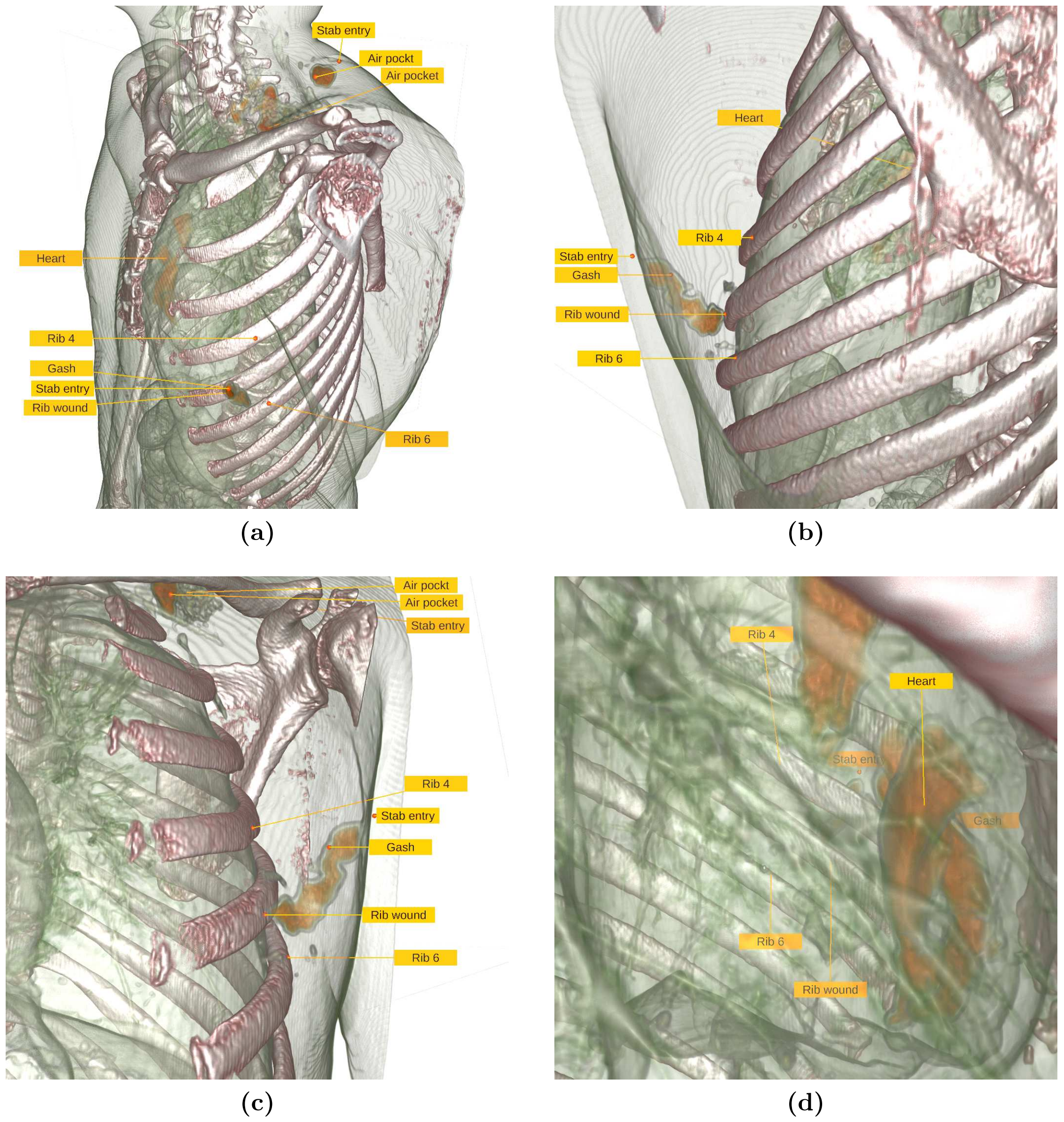


**Figure 5.3:** Multiple stabbing wounds with extracted datasets for air cavities.

### 5.1.4 Heidelberg Fall

Another Heidelberg case is from a man who broke his neck after a high fall. The images are shown in Figure 5.4. Labels are placed on the background wherever possible because of the integral costs in equation (3.1). “T1” is placed onto homogeneous areas in both cases.

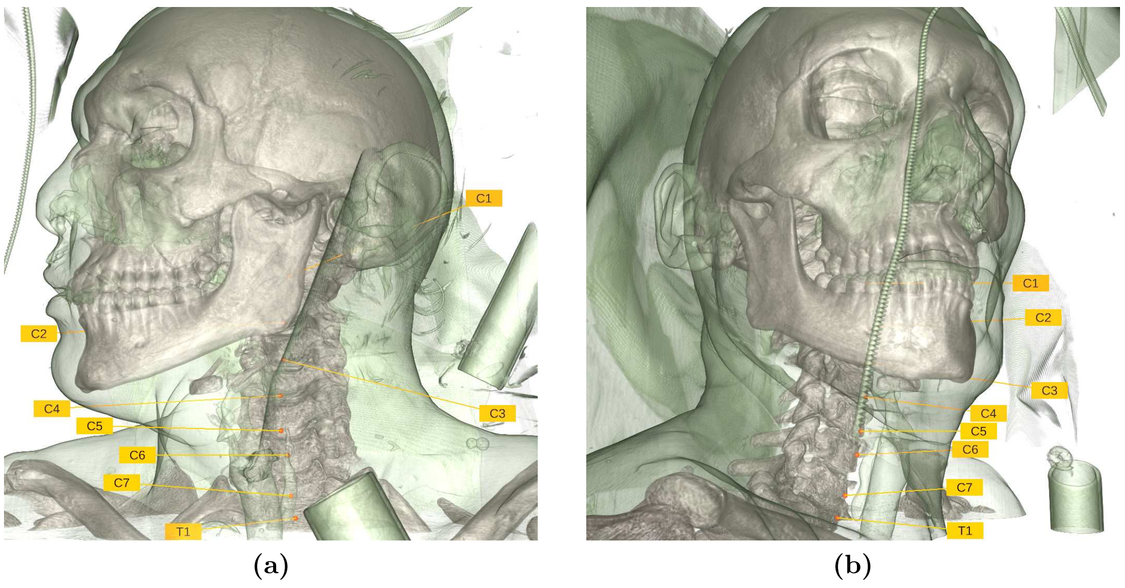


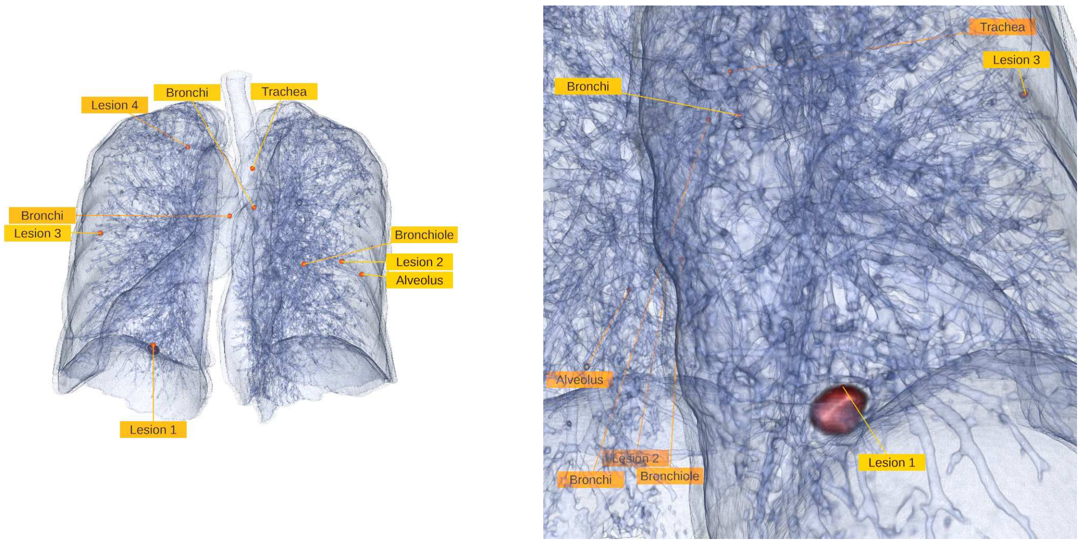
**Figure 5.4:** Dataset of a man with a broken neck inside a body bag. All vertebrae are labelled.

### 5.1.5 LIDC-IDRI Lung

The data for this scene stems from the Lung Image Database Consortium and Image Database Resource Initiative by Armato et al. [1]. They have created a database for lung nodules on CT scans, as described by Armato et al. [2]. A lung nodule is a small growth in the lung, which most of the time is benign, although they can also be cancerous. The used dataset is `1.3.6.1.4.1.14519.5.2.1.6279.6001.558286136379689377915919180358`. The larger nodules are segmented, highlighted and annotated. Examples are given in Figure 5.5 and 5.12b.

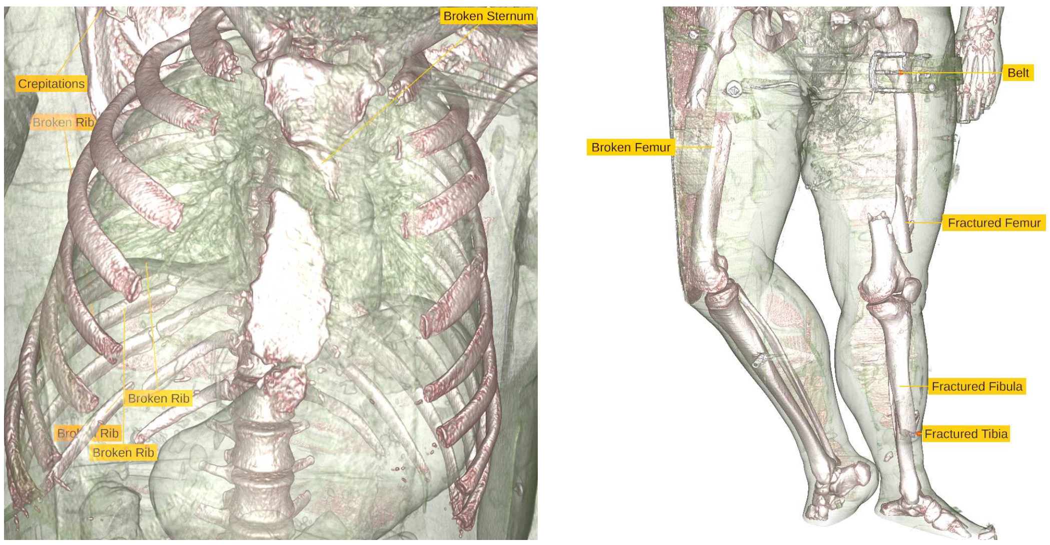
### 5.1.6 Heidelberg Motorcycle

This is a scene of a motorcycle accident. Figure 5.6a shows the torso with broken ribs, broken sternum and a fractured shoulder. One of the broken rib labels has not enough space and is partially occluded. We come back to this particular label in Section 6. The lower body is shown in Figure 5.6b, including broken femurs, fractured tibia and fibula.



**(a)** **(b)**

**Figure 5.5:** Lung dataset with annotated nodules.



**(a)** **(b)**

**Figure 5.6:** CT scan of motorcycle accident victim.

### 5.1.7 Crytek Sponza

To show that the algorithm also works with 3D scenes, the Crytek Sponza model adapted by McGuire[32] is used. Transparency has been added to the curtains to utilize the HA-Buffer and showcase labels behind transparent objects. Results are given in Figure 5.7.

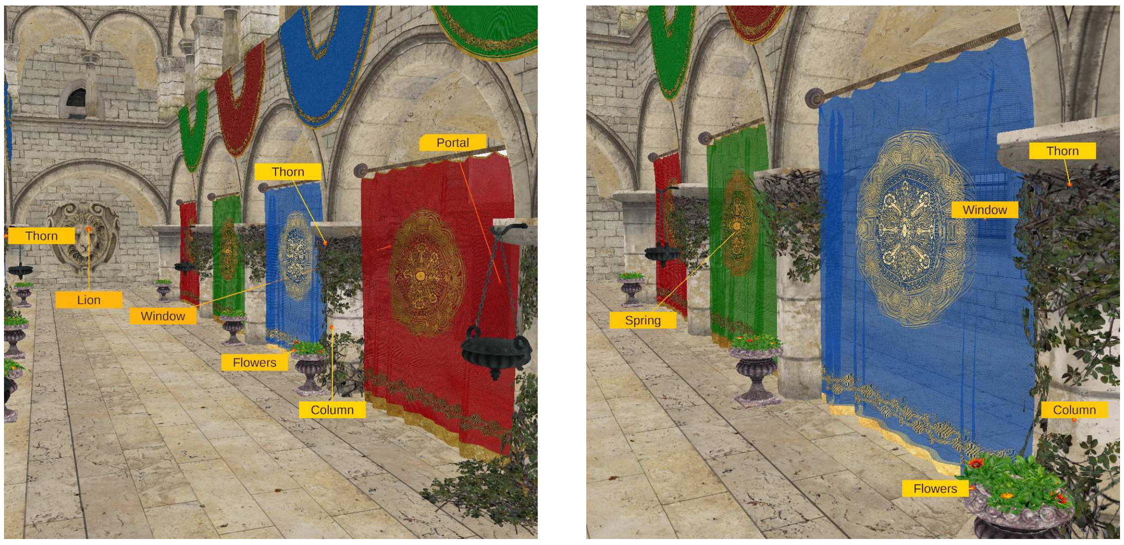


**Figure 5.7:** Screenshots of Sponza scene.

### 5.1.8 Plane

As demonstration of the external labeling capabilities, a 3D plane model is used, see Figure 5.8. The model has been created by Peter Mohr from the Institute for Computer Graphics and Vision and kindly allowed its usage for this work.

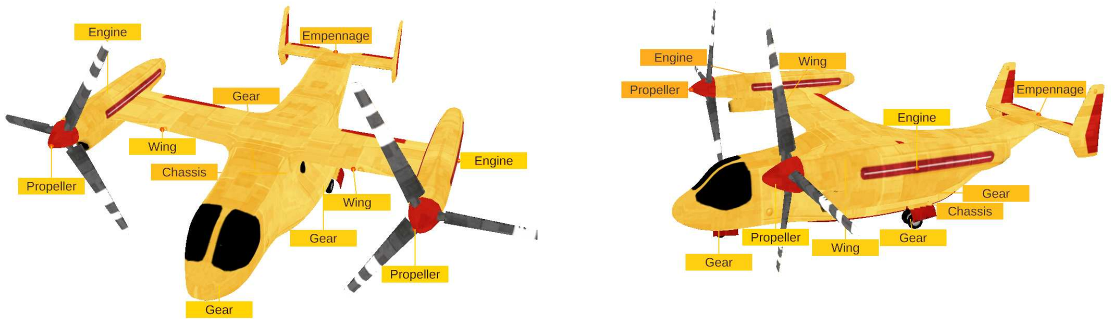
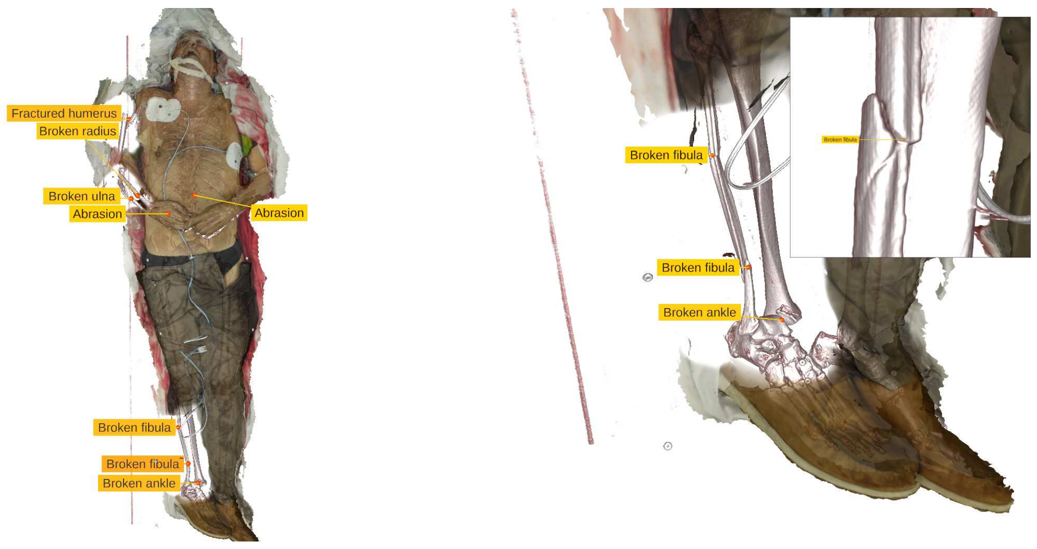


**Figure 5.8:** Different views of an annotated plane model.

### 5.1.9 Pedestrian

This example shows a pedestrian, who has been hit by a car (Figure 5.9). It contains a mesh, reconstructed from photographs of the accident location. The mesh is placed on top of the CT scan. Regions with pathologies in the bones are transparent in the mesh. The upper broken fibula (also in the closeup) has a characteristic triangular shape, which is an indication of the direction of the force that caused the bone to break. The underlying volume dataset was provided by Ludwig Boltzmann Institute for Clinical Forensic Imaging.



(a) (b)

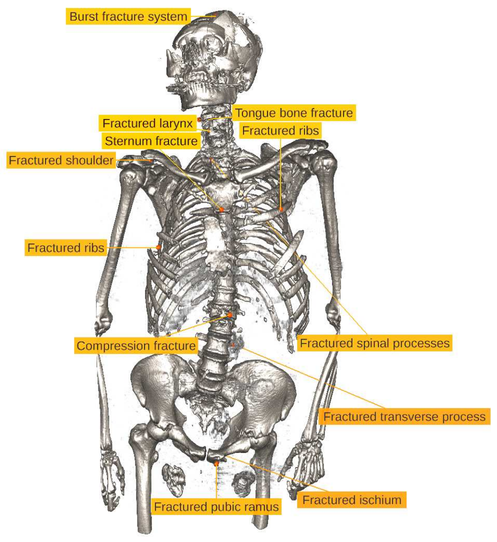
**Figure 5.9:** Combination of a mesh overlay of the crime scene with a CT-Scan.

### 5.1.10 Bus victim

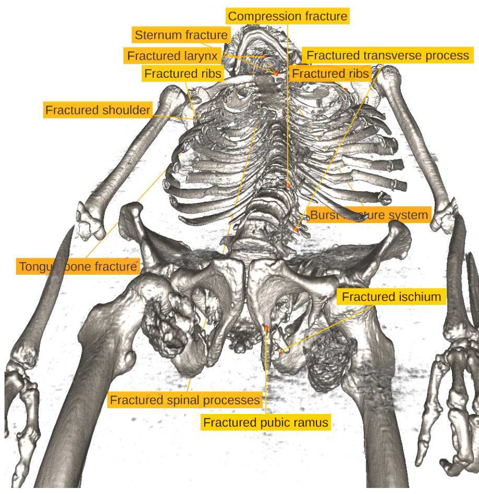
In this forensic case (Figure 5.10 and 5.12a), a man was overrun by a bus and sustained many heavy and lethal injuries. Nearly all labels are rather long, so it is quite hard to find good positions. This dataset was also provided by Ludwig Boltzmann Institute for Clinical Forensic Imaging.

### 5.1.11 Jet engine

Figure 5.11 displays two views of a CAD model of a turbofan jet engine. Labeling CAD models presents another use case, which can be handled by the proposed algorithm. The model was taken from `https://grabcad.com/library/turbo_fan_jet_engine-1`. The right image (Figure 5.11b) shows that the algorithm looks for homogeneous areas to place the labels if no background is available.

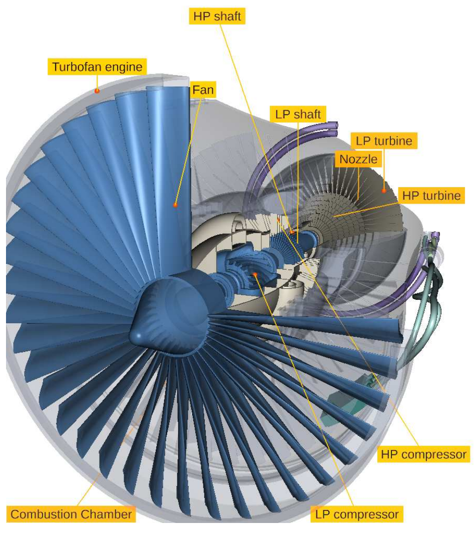


(a)

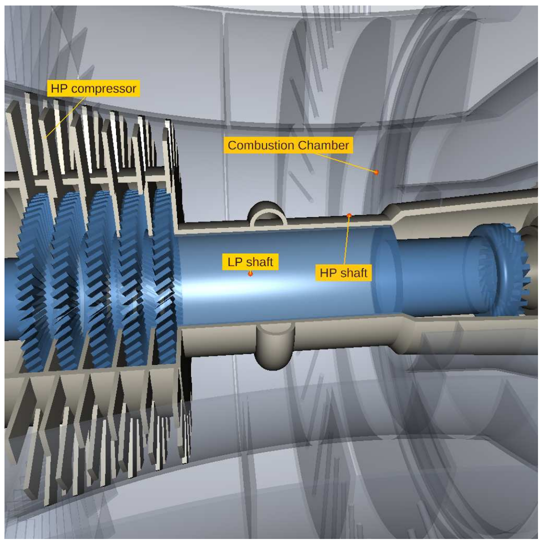


(b)

**Figure 5.10:** Case of a man overrun by a bus. Many long labels are included to challenge the labeling system.



(a)



(b)

**Figure 5.11:** CAD model of a jet engine.

### 5.1.12 100 Labels

In this scene, 100 labels have been randomly positioned and are displayed with the Manix dataset, taken from the Osirix DICOM image library (`http://www.osirix-viewer.com/resources/dicom-image-library/`). An image is shown in Figure 5.14a.

## 5.2 Comparison

Figure 5.12 shows the general differences to the reimplementation of the Stein & Décoret algorithm. The only differences are the insertion order, the layer count and therefore the integral costs (see Section 3.2.2).

Our approach in Figure 5.12a has one severe occlusion, but in Figure 5.12d one label cannot be placed. In Figures 5.12b and 5.12c the labels are closer to the anchor. Some labels further back are partially occluded by details in the foreground, but still remain legible.

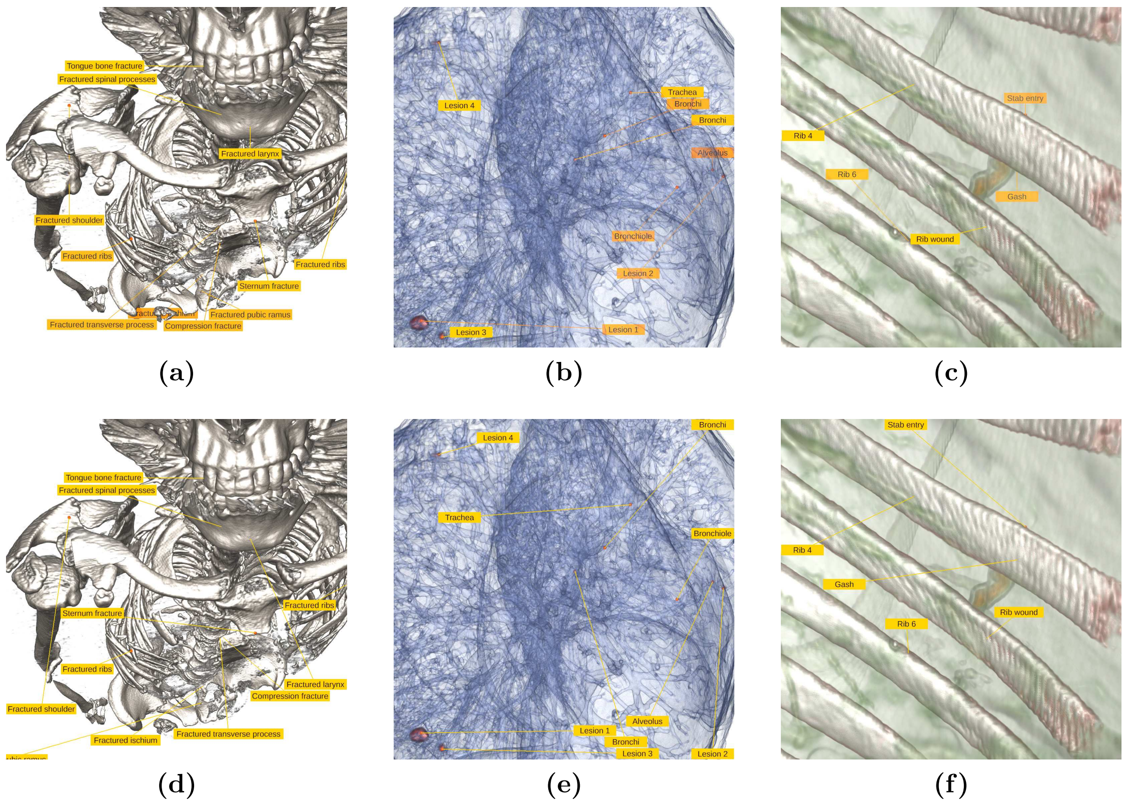


**Figure 5.12:** Images showing general differences between the proposed algorithm in the first row to the Stein & Décoret reimplementation in the second row.

## 5.3 Temporal Behavior

To prevent drastically different placement results during motion, the weight for the distance to the anchor is increased, see Section 3.2. This not only causes smoother label position changes, but also aids the user in navigating complex scenes, since the label is very close to the anchor. An image sequence taken from a video is shown in Figure 5.13.

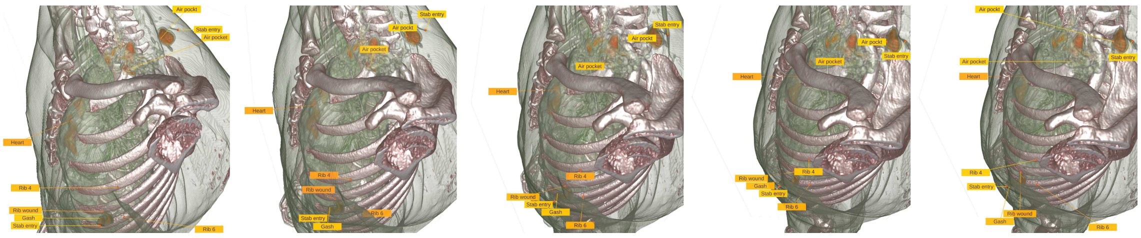


**Figure 5.13:** Placement behavior during camera movement. Labels are stronger attracted to the anchor, for a better temporal coherence and easier navigation.

| # | Scene Name | Labels | Placement | Update | Clustering |
|---|---|---|---|---|---|
| | | - | ms | ms | ms |
| 1 | Knife | 9 | 3.84 | 2.54 | 0.05 |
| 2 | | | 11.46 | 2.42 | 0.04 |
| 3 | Shot | 9 | 4.87 | 2.28 | 0.06 |
| 4 | | | 11.73 | 2.18 | 0.04 |
| 5 | Sponza | 10 | 2.69 | 2.29 | 0.07 |
| 6 | | | 8.55 | 2.31 | 0.07 |
| 7 | Dice | 22 | 10.75 | 2.54 | 0.10 |
| 8 | | | 23.92 | 2.54 | 0.08 |
| 9 | 100 Labels | 100 | 70.86 | 7.96 | 0.38 |
| 10 | | | 179.37 | 8.62 | 0.33 |

**Table 5.2:** Performance measurements of some test scenes. Odd rows are taken from the proposed algorithm with four layers. Even rows are derived from the Stein & Décoret reimplementation using a Apollonius diagram and "hard" constraints.

## 5.4 Performance

The overall labeling performance is determined by the number of labels. Table 5.2 contains performance measurements, which show the exponential complexity. Odd rows contain our results, even rows the timings of Stein & Décoret. The "placement" part primarily entails cost function creation (including the creation of the constraints) and minimization (the "Placement" box in Figure 3.1), "Update" contains calculating and applying the forces, as well as updating all label nodes and distributing labels to layers. "Clustering" is the time to recalculate the cluster centers and transform the labels (therefore the time when using Stein & Décoret is not significantly lower).

All performance measurements were conducted on a PC with an Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz with 4 cores, 16GB RAM and a NVidia GeForce GTX 1070 graphics card with 8GB RAM.

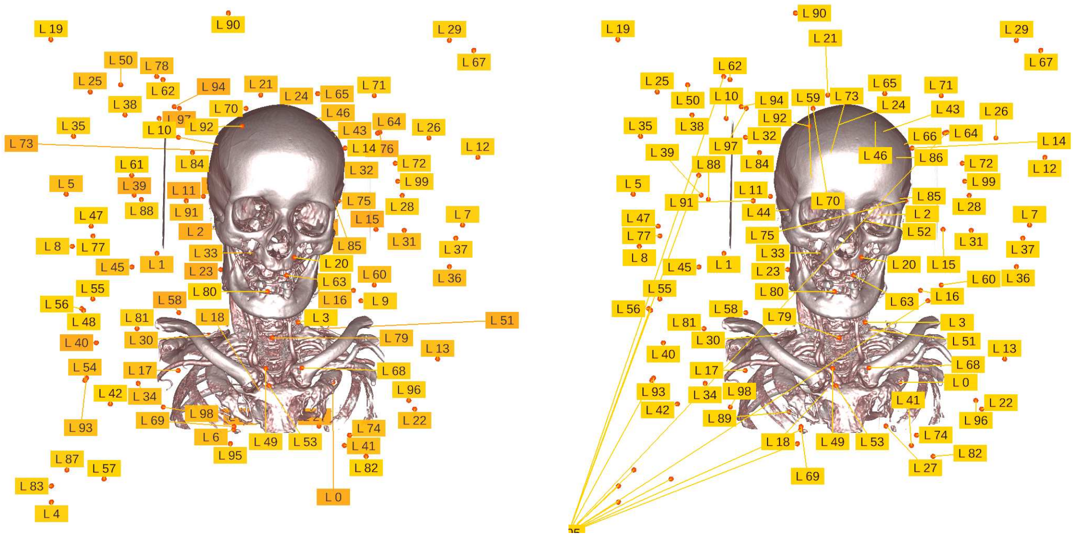


**Figure 5.14:** Stress test using 100 labels placed randomly. (a) shows our result wherein some labels in the rear are occluded. (b) is the result using Stein & Décoret. It cannot place 11 labels, because of the “hard” constraints (moved to the bottom left corner).

# 6 Discussion

## 6.1 Placement

We propose a new labeling algorithm developed for volumetric scenes and scenes with high depth complexity. Our main contribution is slicing the scene into layers. First, to generate a good insertion order for the labels into the greedy placement algorithm, but also to approximate the costs in the third dimension as good as possible for real-time applications and to add depth cues.

One such depth cue is the occluded broken rib label on the left side at the bottom in Figure 5.6a. One could argue that this is a bad placement result, which is certainly true. But since the label is in the last layer there are no good vacant options remaining. The curios user would rotate the camera in any case to get a better view at the broken rib itself. So this label, while being not at a good position, at least guides the user to a better view.

### 6.1.1 Comparison to Stein & Décoret

Our labeling order yields comparable results to the proposed order of Stein & Décoret, which uses the innermost labels first. Differences are primarily a matter of personal preferences and need to be further evaluated, see Figure 5.12. The dissimilarities are also quite certainly attenuated by our reimplementation of the Stein & Décoret algorithm. In essence, the only distinction in our comparison is the different insertion order and the layering. For a single layer the integral costs are also different, as shown in Section 3.2.2. It would be quite interesting to compare results with the original implementation, using the user specified values for the importance of surfaces. The differences can further be minimized by choosing appropriate weights. The current weights used in our approach place labels closer to the anchor at the cost of partial occlusions. The labels still remain readable most of the time and the occlusions add depth cues, which might lead to an overall better comprehensibility of the scene.

The layering approach is faster (see Table 5.2) and performs better for higher depth complexities. This is because the insertion order is based on the depth and so labels in the front are placed first and therefore at better positions than labels farther back. Furthermore, the temporal coherence can benefit, because of less frequent changes in the insertion order. Up to seven layers are supported, because of OpenGL render target restrictions. In practice four layers strike a good balance between accuracy and performance.

A further advantage is that we do not require any data, predefined by the user. All heuristics are derived from intermediate or final images, the dataset and 3D scene information at runtime.

### 6.1.2 Temporal coherency

The temporal behavior is hinted at in Figure 5.13. To prevent drastically changing placement results during motion, the weight for the distance to the anchor is increased. This not only causes smoother label position changes, but also aids the user in navigating complex scenes, because the label is very close to the anchor. Many experiments with different weights, also depending on the state of the camera, have been performed. These experiments have resulted in not using the distance to the old position, since it did not improve the temporal coherency.

### 6.1.3 Forces

The first design also included more types of forces, like forces towards the image center and the anchor as well as forces to prevent connector crossings and label collisions as described in [12]. These forces should have prevented undesired temporary label positions during motion, but since the motion is quite fast, they only complicated the label paths and distracted the user. The forces are still implemented and can be activated for other evaluations.

### 6.1.4 Performance

Overall the performance has been optimized, at least for critical parts, from the start by leveraging the power of today's GPUs. Good implementations and references regarding the steps for labeling have been adapted from the `VolyRenderer` [52]. The downside is, that we have no comparison to more naïve implementations. Whether the conversion to CIELAB warrants the runtime cost must be further investigated. Maybe a simple conversion of RGB to gray is sufficient, see Section 7.

The slowest part of the label placement subsystem in our implementation has long been the construction of the constraints. This was first implemented using `Boost Polygon` [45] and later with `Clipper` `http://www.angusj.com/delphi/clipper.php`. Both are optimized implementations on the CPU, that support more complex dilation operations than necessary and have many more features. Using a geometry shader implementation of the adapted conservative rasterization algorithm as described in Section 4.1 has yielded better performance with a smaller codebase and without any additional external dependencies.

### 6.1.5 Goals

All implementation goals, except for two, have been met. We had the goal to perform the cost function minimization asynchronously, but due to tight coupling with the rendering part, this would have been difficult. Also, it would have required more memory to cache the intermediate results. Copying them from the render targets would probably have nullified the performance gain of the asynchronous calculation. So, this goal has become obsolete. The second unmet goal is adjusting the blending of the layers by increasing the transparency for regions that would occlude labels in layers further behind. This has not been implemented because of time constraints and because the results are good enough without this modification.

## 6.2 Rendering

### 6.2.1 HA-Buffer

The HA-Buffer works great in combination with volume rendering, because it is quite trivial to handle the volume rendering part. For very complex scenes, when the HA-Buffer overflows, strange artifacts are generated. After increasing the default size to $5793 \times 5793$, no new test scene has lead to artifacts. The given buffer size results in a GPU memory usage of nearly 1.5GB (as explained in Section 4.3). This is quite a lot and could be optimized by choosing smaller datatypes for the color or the identifier at the cost of precision and the number of elements in the scene, respectively. Also, the buffer size could be adapted at runtime so that only the necessary memory is allocated, as proposed by Lefebvre et al. [22].

A buffer with this size can manage on average a depth complexity of six fragments stacked behind each other for each pixel on the screen. This might seem high, but consider a scene with two volumes. One could be a detailed head and the other the body. Now, if the view is inside the head and looks down through the body we already need 4 fragments, for the volume boundaries. Additional meshes or another volume would already be sufficient to fill the buffer or cause an overflow.

To prevent overflow artifacts, an early cull mechanism, as described by Lefebvre et al. [22], could be implemented. Then the farthest transparent pixels would be dropped.

### 6.2.2 Layering

Currently, each layer requires its own render target, and one target is necessary for the accumulated result image. This could be optimized so that the back buffer is used for the accumulated result image.

### 6.2.3 Volume Rendering

Volume rendering is the bottleneck in most situations. Producing intermediate results for the layers has no measurable runtime penalty, just increased memory requirements. This may be due to better internal synchronization, when writing results to the layers. Overall, the current implementation causes quite severe branch divergence. This was not addressed, because it would make the code much more complicated. Other ideas to increase the performance are given in the next section.

# 7 Future Work

To further communicate important findings to the user, we would like to add labels with attached photographs. This would be particularly useful for superficial injuries. Since labels can have an arbitrary size, only the label rendering has to be adjusted, the labeling part already supports everything needed for labels with images.

The label movement could probably be smoothed by using a more complex filter, like the Kalman filter. Easing curves could be applied for a more natural feel, and also to increase the maximum speed.

In general the limitation of the system is currently the performance of the volume renderer. There are many possible ways to increase its performance, like empty space skipping [21] or marching cubes [27] and extensions thereof.

The volume renderer could also be extended with multidimensional transfer functions, CSG and more advanced lighting models as described by Preim and Botha [39].

Finally, we would like to conduct a user study to validate our approach and further receive feedback. The following evaluations could be performed:

- use different cost function term weights
- changing the weights depending on the system state
- add different transitions between the states
- experiment with different overall force strength
- try to include the other forces for certain situations, for example, include them if the target position is far away
- evaluate different depth cues and evaluate how much they enhance the depth perception
- different implementations of the saliency, for example with a simple conversion to a grayscale image

# A Application

The source code for the application is available at `https://github.com/Christof/voly-labeller`. Listing 3 shows the help output of the application.

```
Usage: ./voly-labeller [options] scene
Multiple labelling implementations for volume rendered medical data

Options:
  -h, --help                          Displays this help.
  -v, --version                       Displays version information.
  --offline                           Enables offline rendering
  --layers <Layer Count>              Number of layers. Default is 4
  --hard-constraints                  Simulate hard constraints
  --apollonius                        Use apollonius graph to determine label insertion order
  --disable-labelling                 Disable drawing lables
  --internal-labelling                Enable internal labelling
  --optimize-on-idle                  Optimize costs when the camera is not moving
  --show-buffers                      Show buffers for debugging
  --decoret                           Change parameters to simulate Stein & Décoret algorithm:
                                      1 layer, using apollonius graph and hard constraints
  -s, --screenshot <Camera Position>  Takes a screenshot of the given camera position. Characters
                                      after a '_' are ignored but added to the filenameMultiple
                                      camera positions for multiple screenshots can be supplied
                                      with ',' as separator. Camera Positions containing a space
                                      character are not supported!
  --video <Camera Positions>          Take a video of the given camera
                                      positions. The camera positions must be
                                      supplied with ',' as separator. After a
                                      '_' the duration to the specified position
                                      can be specified in seconds.
  --movement <Camera Positions>       Move to the specified camera positions.
                                      The camera positions must be supplied with
                                      ',' as separator. After a '_' the duration
                                      to the specified position can be specified
                                      in seconds.

Arguments:
  scene                               Scene file to load.
```

**Listing 3:** Application help

# Bibliography

[1] Armato, S. G., McLennan, G., Hawkins, D., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., Hoffman, E. a., Kazerooni, E. a., MacMahon, H., Van Beeke, E. J. R., Yankelevitz, D., Biancardi, A. M., Bland, P. H., Brown, M. S., Engelmann, R. M., Laderach, G. E., Max, D., Pais, R. C., Qing, D. P. Y., Roberts, R. Y., Smith, A. R., Starkey, A., Batrah, P., Caligiuri, P., Farooqi, A., Gladish, G. W., Jude, C. M., Munden, R. F., Petkovska, I., Quint, L. E., Schwartz, L. H., Sundaram, B., Dodd, L. E., Fenimore, C., Gur, D., Petrick, N., Freymann, J., Kirby, J., Hughes, B., Casteele, A. V., Gupte, S., Sallamm, M., Heath, M. D., Kuhn, M. H., Dharaiya, E., Burns, R., Fryd, D. S., Salganicoff, M., Anand, V., Shreter, U., Vastagh, S., Croft, B. Y., and Clarke, L. P. (2015). Data From LIDC-IDRI. The Cancer Imaging Archive. (page 27)

[2] Armato, S. G., McLennan, G., Hawkins, D., Bidaut, L., McNitt-Gray, M. F., Meyer, C. R., Reeves, A. P., Zhao, B., Aberle, D. R., Henschke, C. I., Hoffman, E. a., Kazerooni, E. a., MacMahon, H., Van Beeke, E. J. R., Yankelevitz, D., Biancardi, A. M., Bland, P. H., Brown, M. S., Engelmann, R. M., Laderach, G. E., Max, D., Pais, R. C., Qing, D. P. Y., Roberts, R. Y., Smith, A. R., Starkey, A., Batrah, P., Caligiuri, P., Farooqi, A., Gladish, G. W., Jude, C. M., Munden, R. F., Petkovska, I., Quint, L. E., Schwartz, L. H., Sundaram, B., Dodd, L. E., Fenimore, C., Gur, D., Petrick, N., Freymann, J., Kirby, J., Hughes, B., Casteele, A. V., Gupte, S., Sallamm, M., Heath, M. D., Kuhn, M. H., Dharaiya, E., Burns, R., Fryd, D. S., Salganicoff, M., Anand, V., Shreter, U., Vastagh, S., Croft, B. Y., and Clarke, L. P. (2011). The Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI): a completed reference database of lung nodules on CT scans. *Med. Phys.*, 38(2):915–931. (page 27)

[3] Arthur, D. and Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding. In *Proc. eighteenth Annu. ACM-SIAM Symp. Discret. algorithms*, volume 8, pages

1027–1035. Society for Industrial and Applied Mathematics Philadelphia, PA, USA. (page 14)

[4] Bell, B., Feiner, S., and Höllerer, T. (2001). View management for virtual and augmented reality. *Proc. 14th Annu. ACM Symp. User interface Softw. Technol. - UIST '01*, 2001:101. (page 5)

[5] Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. *ACM SIGGRAPH Comput. Graph.*, 11(2):192–198. (page 19)

[6] Bruckner, S. and Gröller, M. E. (2006). Exploded views for volume data. *IEEE Trans. Vis. Comput. Graph.*, 12(5):1077–1084. (page 4)

[7] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-8(6):679–698. (page 11)

[8] Christensen, J., Marks, J., and Shieber, S. (1995). An empirical study of algorithms for point-feature label placement. *ACM Trans. Graph.*, 14:203–232. (page 5)

[9] Flato, E. and Halperin, D. (2000). Robust and Efficient Construction of Planar Minkowski Sums. In *Abstr. 16th Eur. Work. Comput. Geom.*, pages 85–88. (page 17)

[10] Gary, M. R. and Johnson, D. S. (1979). Computers and intractability: A guide to the theory of np-completeness. (page 4)

[11] Grasset, R., Langlotz, T., Kalkofen, D., Tatzgern, M., and Schmalstieg, D. (2012). Image-driven view management for augmented reality browsers. In *ISMAR 2012 - 11th IEEE Int. Symp. Mix. Augment. Real. 2012, Sci. Technol. Pap.*, pages 177–186. (page 5, 11)

[12] Hartmann, K., Ali, K., and Strothotte, T. (2004). Floating labels: Applying dynamic potential fields for label layout. *Smart Graph.*, pages 101–113. (page 5, 15, 36)

[13] Hasselgren, J., Akenine-Möller, T., and Ohlsson, L. (2005). Conservative rasterization. *GPU Gems*, 2:677–690. (page viii, 17, 18)

[14] Höhne, K. H. and Bernstein, R. (1986). Shading 3D- Images from CT Using Gray-Level Gradients. *IEEE Trans. Med. Imaging*, MI-5(1):45–47. (page 19)

[15] Imhof, E. (1975). Positioning Names on Maps. *Cartogr. Geogr. Inf. Sci.*, 2(2):128–144. (page 4, 8)

[16] Jain, A. K. (1989). *Fundamentals of Digital Image Processing.* Prentice-Hall, Inc. (page 11)

[17] Kainz, B., Grabner, M., Bornik, A., Hauswiesner, S., Muehl, J., and Schmalstieg, D. (2009). Ray casting of multiple volumetric datasets with polyhedral boundaries on manycore GPUs. *ACM Trans. Graph.*, 28(5):1. (page 3, 16)

[18] Karavelas, M. I. and Yvinec, M. (2002). Dynamic additively weighted Voronoi diagrams in 2D. *Algorithms-Esa 2002, Proc.*, 2461:586–598. (page 5)

[19] Koller, J. (2016). *Image Shading Corretion via TGV and DCT with Appliation to MRI*. PhD thesis, Graz University of Technology. (page 23)

[20] Krüger, J., Schneider, J., and Westermann, R. (2006). ClearView: An interactive context preserving hotspot visualization technique. *IEEE Trans. Vis. Comput. Graph.*, 12(5):941–948. (page 4)

[21] Kruger, J. and Westermann, R. (2003). Acceleration techniques for GPU-based volume rendering. *IEEE Vis. 2003. VIS 2003.* (page 3, 19, 39)

[22] Lefebvre, S., Hornus, S., and Lasram, A. (2013). HA-Buffer: Coherent Hashing for single-pass A-buffer. Technical Report April, INRIA. (page 3, 37)

[23] Lefebvre, S., Hornus, S., and Lasram, A. (2014). Per-Pixel Lists for Single Pass A-Buffer. *GPU Pro 5 Adv. Render. Tech.* (page 3, 19)

[24] Levoy, M. (1988). Display of surfaces from volume data. *IEEE Comput. Graph. Appl.*, 8(3):29–37. (page 19)

[25] Levoy, M. (1990). Efficient ray tracing of volume data. *ACM Trans. Graph.*, 9(3):245–261. (page 19)

[26] Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–137. (page 14)

[27] Lorensen, W. E. W. and Cline, H. H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm. *ACM Siggraph Comput. Graph.*, 21(4):163–169. (page 39)

[28] MacQueen, J. B. (1967). Kmeans Some Methods for classification and Analysis of Multivariate Observations. *5th Berkeley Symp. Math. Stat. Probab. 1967*, 1(233):281–297. (page 14)

[29] Mallinckrodt Institute of Radiology Washington University School of Medicine (2010). Whole body morphometry project. (page 23)

[30] Marks, J. and Shieber, S. (1993). The Computational Complexity of Cartographic Label Placement. *English*. (page 1, 4)

[31] Maule, M., Comba, J. L. D., Torchelsen, R. P., and Bastos, R. (2011). A survey of raster-based transparency techniques. *Comput. Graph.*, 35(6):1023–1034. (page 3)

[32] McGuire, M. (2011). Computer Graphics Archive. (page 28)

[33] Mogalle, K., Tietjen, C., Soza, G., and Preim, B. (2012). Constrained Labeling of 2D Slice Data for Reading Images in Radiology. In Ropinski, T., Ynnerman, A., Botha, C., and Roerdink, J., editors, *Eurographics Work. Vis. Comput. Biol. Med.*, pages 131–138. The Eurographics Association. (page 5)

[34] Nehab, D., Maximo, A., Lima, R. S., and Hoppe, H. (2011). GPU-efficient recursive filtering and summed-area tables. *ACM Trans. Graph.*, 30(6):1. (page 18)

[35] Oeltze-Jafra, S. and Preim, B. (2014). Survey of Labeling Techniques in Medical Visualizations. In *Proc. 4th Eurographics Work. Vis. Comput. Biol. Med.*, pages 199—-208, Vienna, Austria. Eurographics Association. (page 4)

[36] Pick, S., Hentschel, B., Tedjo-Palczynski, I., Wolter, M., and Kuhlen, T. (2010). Automated Positioning of Annotations in Immersive Virtual Environments. *EGVE - JVRC'10 Proc. 16th Eurographics Conf. Virtual Environ. Second Jt. Virtual Real.*, pages 1–8. (page 5)

[37] Porter, T. and Duff, T. (1984). Compositing digital images. *ACM SIGGRAPH Comput. Graph.*, 18(3):253–259. (page 16)

[38] Preim, B. (1997). *Interaktive Illustrationen und Animationen zur Erklärung komplexer räumlicher Zusammenhänge.* PhD thesis, Otto-von-Guericke-Universität Magdeburg. (page 4)

[39] Preim, B. and Botha, C. P. (2013). *Visual Computing for Medicine: Theory, Algorithms, and Applications.* Newnes. (page 3, 39)

[40] Preuß, M., Schwefel, H.-P., and Kursawe, F. (1998). *Solving Map Labeling Problems by Means of Evolution Strategies.* PhD thesis, Universität Dortmund. (page 5)

[41] Rong, G. and Tan, T.-s. (2006). Jump flooding in GPU with applications to Voronoi diagram and distance transform. *Stud. Log. Theory, Am. Philos. Q. Monogr. 2*, pages 109–116. (page 12)

[42] Ropinski, T., Praßni, J.-S., Roters, J., and Hinrichs, K. H. (2007). Internal Labels as Shape Cues for Medical Illustration. In *Proc. 12th Int. Fall Work. Vision, Model. Vis.*, pages 203–212. (page 4)

[43] Schanda, J. (2007). *Colorimetry: understanding the CIE system.* John Wiley & Sons. (page 11)

[44] Silva, C. T., Comba, J. L. D., Callahan, S. P., and Bernardon, F. F. (2005). A Survey of GPU-Based Volume Rendering of Unstructured Grids Abstract. *Processing*, 31(1):1–22. (page 3, 16)

[45] Simonson, L. and Suto, G. (2009). Geometry Template Library for STL-like 2D Operations. *Color. GTL Boost.* (page 36)

[46] Sobel, I. (1990). An isotropic 3 by 3 image gradient operator. *Mach. Vis. three-demensional Sci.*, 1(1):23–34. (page 11)

[47] Stadler, G., Steiner, T., and Beiglböck, J. (2006). A Practical Map Labeling Algorithm Utilizing Morphological Image Processing and Force-directed Methods. (page 5)

[48] Stein, T. and Décoret, X. (2008). Dynamic Label Placement for Improved Interactive Exploration. In *Proc. Sixth Int. Symp. Non-Photorealistic Animat. Render. (NPAR 2008, June 9–11, 2008, Annecy, Fr.*, pages 15–21. (page 1, 5, 8, 9, 12)

[49] Steinhaus, H. (1956). Sur la division des corp materiels en parties. *Bull. Acad. Pol. Sci. IV*, IV(12):801–804. (page 14)

[50] Tatzgern, M., Kalkofen, D., Grasset, R., and Schmalstieg, D. (2014). Hedgehog labeling: View management techniques for external labels in 3D space. *Proc. - IEEE Virtual Real.*, pages 27–32. (page 5)

[51] Tkalčič, M. and Tasič, J. F. (2003). Colour spaces - Perceptual, historical and applicational background. *IEEE Reg. 8 EUROCON 2003 Comput. as a Tool - Proc.*, A:304–308. (page 11)

[52] Urschler, M., Bornik, A., Scheurer, E., Yen, K., Bischof, H., and Schmalstieg, D. (2012). Forensic-case analysis: from 3D imaging to interactive visualization. *IEEE Comput. Graph. Appl.*, 32(4):79–87. (page 1, 5, 36)

[53] Vaaraniemi, M., Treib, M., and Westermann, R. (2012). Temporally coherent real-time labeling of dynamic scenes. In *Proc. 3rd Int. Conf. Comput. Geospatial Res. Appl. - COM.Geo '12*, page 1. (page 5)

[54] Vasilakis, A. A. and Fudos, I. (2013). Depth-fighting aware methods for multifragment rendering. *IEEE Trans. Vis. Comput. Graph.*, 19(6):967–977. (page 18)

[55] Wein, R. (2006). Exact and efficient construction of planar Minkowski sums using the convolution method. *Algorithms–ESA 2006*, 006413:1–12. (page 17)

[56] Yoeli, P. (1972). The Logic of Automated Map Lettering. (page 4)