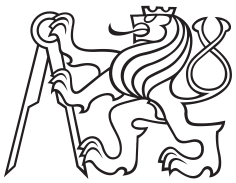
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Mixed labeling: Integration of internal and external labeling

Bc. Václav Pavlovec

Supervisor: Ing. Ladislav Čmolík, Ph.D.
May 2019

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pavlovec** Jméno: **Václav** Osobní číslo: **378929**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Interakce člověka s počítačem**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Integrace interních a externích popisků

Název diplomové práce anglicky:

Mixed labeling: Integration of internal and external labeling

Pokyny pro vypracování:

Analyzujte algoritmy pro automatické rozmisťování interních a externích popisků pro 2D a 3D scény. Na základě analýzy navrhněte a implementujte algoritmus, který umožní pro 2D a 3D scény kombinovat interní a externí popisky. Externí popisky se snažte umístit tak, aby bylo možné přiřadit popisky ke správným objektům i bez vodících čar. Výslednou implementaci otestujte alespoň na pěti scénách různé složitosti obsahujících alespoň 10 popisovaných objektů. Schopnost algoritmu rozmisťovat popisky tak, že je možné přiřadit popisky k popisovaným objektům, vyhodnoťte pomocí uživatelské studie.

Seznam doporučené literatury:

Kouřil D., L. Čmolík, B. Kozlíková, H. Wu, G. Johnson, D. S. Goodsell, A. Olson, M. E. Gröller, and I. Viola. Labels on Levels: Labeling of Multi-Scale Multi-Instance and Crowded 3D Biological Environments, IEEE Transactions on Visualization and Computer Graphics, 25(1):977-986, 2019.
Čmolík L. and J. Bittner. Real-time External Labeling of Ghosted Views. IEEE Transactions on Visualization and Computer Graphics (Early access), 2018.
Čmolík L. and J. Bittner. Layout-aware optimization for interactive labeling of 3d models. Computers & Graphics, 34(4):378-387, 2010.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ladislav Čmolík, Ph.D., Katedra počítačové grafiky a interakce

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **14.02.2019** Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **20.09.2020**

[redacted]
Ing. Ladislav Čmolík, Ph.D.
podpis vedoucí(ho) práce

[redacted]
podpis vedoucí(ho) ústavu/katedry

[redacted]
prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

Datum převzetí zadání

[redacted]
Podpis studenta

Acknowledgements

I wish to express my deepest gratitude to Ing. Ladislav Čmolík, Ph.D. for his guidance and valuable consultations. I would also like to thank my family and my friends for their unwavering support.

Declaration

I hereby declare that I created the presented thesis on my own, and I quoted all used sources of information, following the Methodical instructions on ethical principles for writing an academic thesis.

Abstract

This master thesis focuses on automatic label placement. Labels can be positioned over a given object (Internal Labeling) or next to it and connected by a line (External Labeling). I attempt to combine the two methods and position the labels in a mixed fashion. When I place the label next to the object, I try to pick such a position, that the line to connect it is not necessary.

First, I present existing methods in the text. Subsequently, I analyze the problems related to mixed label placement. I carry out the implementation based on the analysis and test it with six models of various complexity. I evaluate with users in the end.

Keywords: automatic label placement, labeling, external labeling, internal labeling, mixed labeling

Supervisor: Ing. Ladislav Čmolík, Ph.D.

Abstrakt

Tato diplomová práce se zabývá automatickým rozmístěním popisků. Popisky mohou být umístěny přes daný objekt (angl. Internal Labelig) či vedle daného objektu (angl. External Labeling), přičemž jsou s ním spojeny čárou. V této práci se pokouším zkombinovat tyto dva přístupy a popisky umísťovat smíšeně. Zároveň se snažím popisky umístěné vně objektu umístit tak, aby k jejich asociaci s ním nebyla čára potřebná.

V textu práce nejprve uvádím existující přístupy k této problematice. Následně analyzuji problémy spojené se smíšeným umísťováním popisků a na základně analýzy provádím implementaci. Funkčnost algoritmu je otestována na šesti modelech. Se třemi z nich je provedena uživatelská studie.

Klíčová slova: automatické umístění popisků, popisky, externí popisky, interní popisky, smíšené popisky

Překlad názvu: Integrace interních a externích popisků

Contents

Figures

Tables

Chapter 1

Introduction

Complex models consisting of a large number of parts appear in a multitude of domains. Illustration of these models conveys information about their spatial arrangement. In order to communicate the function of individual components, these models comprise of, a textual description is required. Labels represent the means of connecting the textual description with the visual representation of these objects.

Based on the position of the labels, we talk about internal, external, or mixed labeling. The internal labels are positioned over the parts, therefore using proximity to communicate relation with a given part of the model. They are commonly used in cartography to mark area features since maps generally utilize all of the available space. Hence the labels must overlap the graphic elements. In contrast, the external labeling methods place labels on the outside of objects, almost exclusively utilizing leader lines to connect them with a given part. There exists a variety of external labeling methods, suitable for various purposes. Applications include boundary labeling of maps, 3D scenes, 3D models, textbook illustrations, and many others that I will describe in more detail in the subsequent chapter. Combination of internal and external labels yields a mixture of the two styles, producing the most general case of labeling - mixed labeling.

Labeling should support various styles in order to efficiently communicate relations between labels and individual parts of the model. According to Tufte [33], labeling should integrate all the necessary information into graphics itself to prevent the user's eye from darting back and forth between the underlying textual material and the graphic.

Label layout, i.e., positioning of the labels, should exhibit several basic properties as described by Ali et al. [4] It should be readable, unambiguous, aesthetically pleasing, real-time, frame-coherent and compact [4]. More precisely, labels should be placed in such a manner, so there is no visual interference, and the user can read them with no difficulties. The positioning of the labels can pose problems pertaining to the association among the labels and the objects. While this is not typically a problem if the labels are placed externally and connected with lines, when this is not the case, significant issues may arise. Since the results are made to be viewed by a user, they should also be aesthetically pleasing, not lessened by the presence of visual

clutter. Labels should not take up any extra space around the illustration. It is especially important if the results are meant to be embedded into a text, as accompanying illustrations.

1.1 Motivation and Contribution

This work can be viewed as a continuation of my bachelor's thesis [26], where I explored the problem of external labeling of 3D objects. Some approaches are carried over from it, meaning that I have more space to focus on innovation, as I am already familiar with a part of the problem. The aim of this thesis is to design an algorithm capable of combining the methods of internal and external labeling. Ability to place labels in a more versatile fashion allows for more efficient usage of space and reduction of visual clutter. Which follows the aforementioned requirements for the label layout, as all of the qualities can be improved when compared to the methods restricted to one type of labels. Possible use cases for this algorithm are rather broad, including labeling of packed scenes, such as the one depicted in Figure 1.1, static illustrations, and 3D models, to name just a few. As the algorithm operates in real time for medium complexity scenes, it is suitable for use in interactive applications, where the user needs swift feedback.

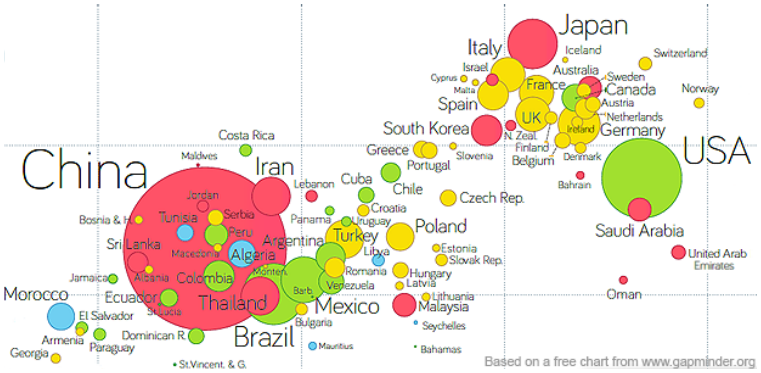


Figure 1.1: Hand created labeling of a chart from gapminder.org. Source[13].

1.2 Structure of the thesis

In chapter 2 of this thesis, I assesses existing methods of both internal and external labeling in order to inspire the subsequent design of the algorithm. In chapter 3, I bring an overview of the algorithm and describe its steps. Subsequently, in chapter 4, I describe the implementation of the algorithm, followed by the implementation details and visualization of the auxiliary data structures the algorithm utilizes. The following chapter 5 asses the results produced by the algorithm and evaluates its performance on scenes of different complexity. I perform a user study in chapter 6 to evaluate the results from a user standpoint. Finally, I summarize the work on this thesis in chapter 7.

Chapter 2

Related work

In this chapter, I describe various methods published over the years. Furthermore, I address the suitability of these methods for the desired algorithm. I divide the discussion based on the positioning of the labels into sections on internal, external, and mixed labeling methods. A substantial part of the literature is concerned with the labeling of point features. While their relevance might not be too high, as my focus lies with the labeling of area features, they are listed here anyway to paint a more holistic picture of the methods zoo.

2.1 Internal labeling

In case that the scene we mean to label contains only disjoint parts able to accommodate the entirety of the label surface, the internal labels are the preferred option. This prerequisite is met in case of maps, although there are quite a few use cases from the field of information visualization. According to Hartman et al. [18], the internal labels can be aligned to a horizontal line, or they can be curved, thus indicating the shape of the labeled object. The internal labeling is most commonly utilized to describe area features in the cartographic domain. The guidelines in this domain have been long established by Yoeli [36]. Research communities developed many approaches to place the labels for maps automatically. Agarwal et al. [2] model the labels as rectangles parallel with horizontal axes. They attempt to find positions of the rectangles so that they do not overlap each other. Freeman [14] presented an algorithm capable of automated cartographic text placement. Freeman attempts to label point, line, and area features following the rules and conventions of the cartographers. Freeman resolves ambiguous label positions based on priorities for the label groups. Figure 2.1 depicts the results obtained by his method.

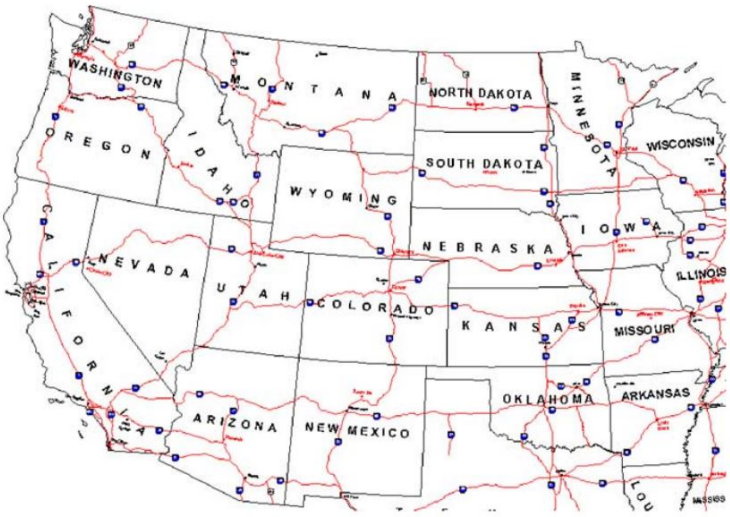


Figure 2.1: Results obtained through the method of Freeman. Source [14].

Further uses of the internal labeling can be found in massive CAD models, as demonstrated by Prado and Raposo [27] or to annotate surfaces in medical illustrations, as shown by Ropinski et al. [29]. Ropinski et al. attempt to provide additional shape cues by fitting the labels into the depth structure of their objects, as depicted in Figure 2.2.

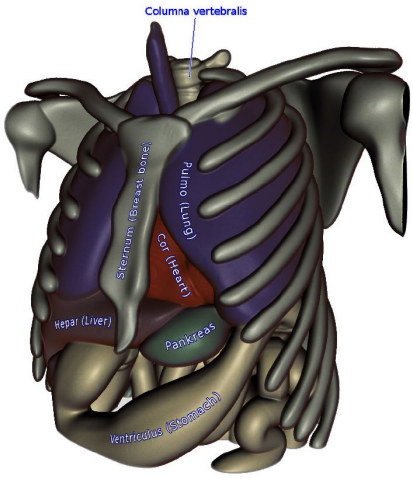


Figure 2.2: Results obtained via the method of Ropinski et al.[29]. Source [29].

Kouřil et al. [21] presented a method of labeling a large number of hierarchically organized features in an interactive 3D environment. The features they label are multi-instance and multi-scale. Thus, the label is not placed for only one feature but it represents all of the instances. The results generated by their method are depicted in Figure 2.3.

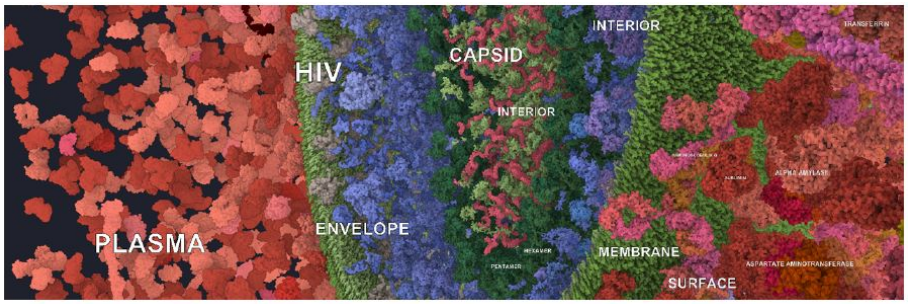


Figure 2.3: Results obtained via the method of Kouřil et al.[21]. Source [21].

2.2 External labeling

External labeling algorithms place the labels close to the objects they mark and connect them with a line segment. A point of the line segment, which connects to the object, is called an anchor, while the line itself is called a leader line. An appropriate anchor must be selected, so it is easily attributable to the object. There are substantial differences among the external labeling algorithms. Thus there arises a need to divide them further. I categorize them into individual sections based on how close to the object they can place the label. Boundary labeling techniques place the labels around the illustrations, as I describe in section 2.2.1. A class of algorithms, which label, predominantly, 3D models, utilizes a projection of the convex hull of the model and places labels at its boundary. I describe these methods in section 2.2.2. Last section 2.2.3 presents algorithms that do not constrain the label positions like the algorithms in the previous sections and can place the labels anywhere in the scene.

2.2.1 Boundary labeling

The method presented by Bekos et al. [5] places the labels around the boundary of a rectangular area. They use several different types of leader lines in this process. There exist one-sided [7] and two-sided versions [24] of the boundary labeling problem. Results obtained via the method of Bekos et al. [5] can be seen in Figure 2.4.

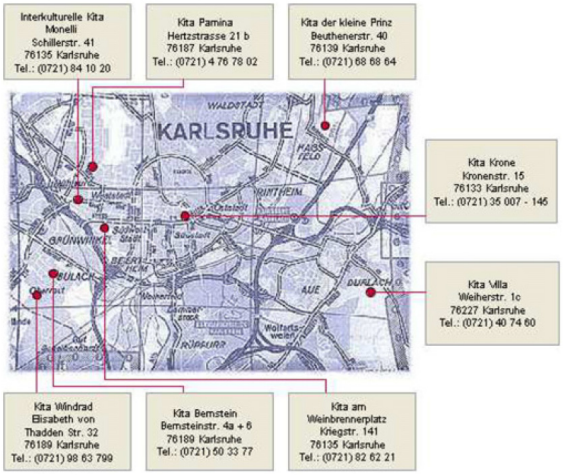


Figure 2.4: Results obtained via the method of Bekos et al. Source [5].

Bekos et al. [24] extended their previous algorithm. In this extended version of the algorithm, the position of the anchors is not fixed, but they are constrained to the predefined rectangular boxes. Results of their approach are depicted in Figure 2.5.

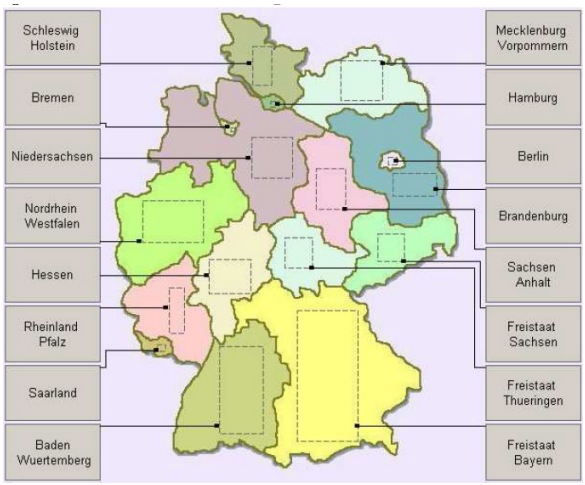


Figure 2.5: Results obtained via the extended method of Bekos et al. Source [24].

2.2.2 Convex labeling

Methods in this section utilize the convex hull of the model and place the labels around it. Ali et al. [4] presented a method that produces the labeling of any given model in real time utilizing its convex silhouette. Positions of anchors are calculated first, and these anchors are subsequently labeled. They support a variety of layout styles, some of which are depicted in Figure 2.6.

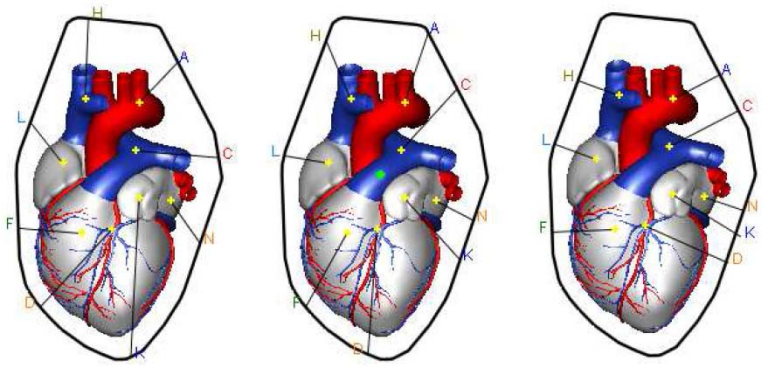


Figure 2.6: Ring, radial, and silhouette-based layout as utilized by the method of Ali et al [4]. Source [4].

The method presented by Čmolík and Bittner [10] also produces real-time labeling. Furthermore, it computes the positions of the anchors and the labels simultaneously. Because of this fact, and employment of a larger number of criteria when evaluating the fitness of leader lines, they obtained aesthetically improved results compared to the method of Ali et al. [4]. The comparison of the results produced by these two methods is depicted in Figure 2.7.

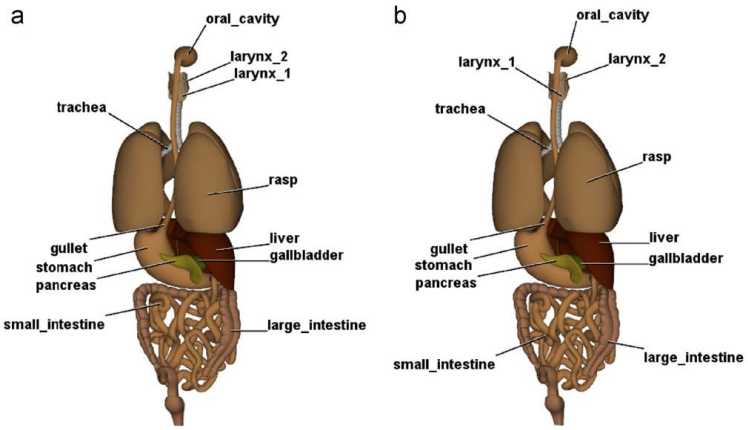


Figure 2.7: Comparison of the methods of Ali et al. (a) and Čmolík and Bittner (b). Source [10].

Čmolík and Bittner [11] presented the algorithm for labeling of ghosted models, as they extended their previous work. They solve the problem by not placing the anchors in the regions where the labeled objects are too transparent or where they are occluded by opaque objects. Thus, the users can correctly associate the label to the corresponding object. Results of their method are depicted in Figure 2.8.

These approaches are partially suitable for the tasks I mean to solve but carry certain limitations imposed on them by the need of convex silhouette.

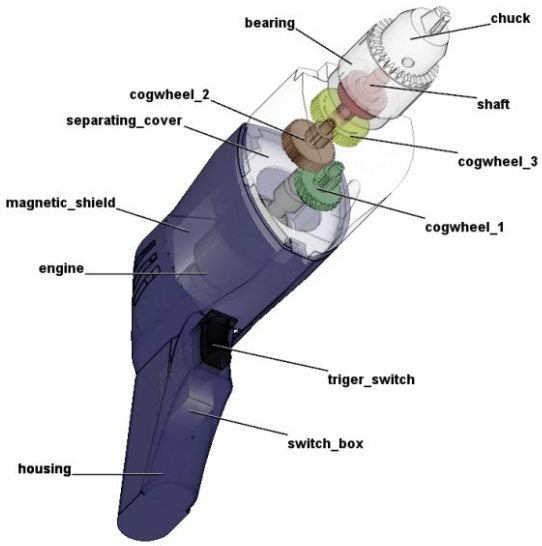


Figure 2.8: Label layout produced by the method of Čmolík and Bittner [11]. Source [11].

2.2.3 Non-Convex labeling

The methods presented in this section can place the labels anywhere in the scene and thus utilize all available free space.

Stein and Décoret [32] presented a greedy algorithm for labeling of a fixed set of anchors attached to 3D objects. Overlap of the labels with the objects is prevented with the utilization of the summed area table and shadow regions. Example of their results is depicted in Figure 2.9. Approach with the summed area table applies to our task, as it enables overlap detection and therefore, its prevention.

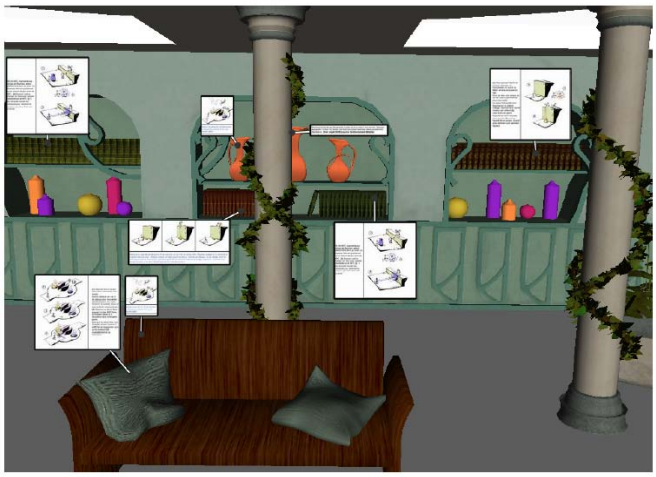


Figure 2.9: Result obtained by Stein and Décoret. Source [32].

Wu et al. [35] presented an algorithm utilizing a zone-based approach for labeling metro maps. The results they obtained are depicted in Figure 2.10. They position the labels in the scene without crossing the metro lines. The labels are connected to the objects with leader lines if necessary.

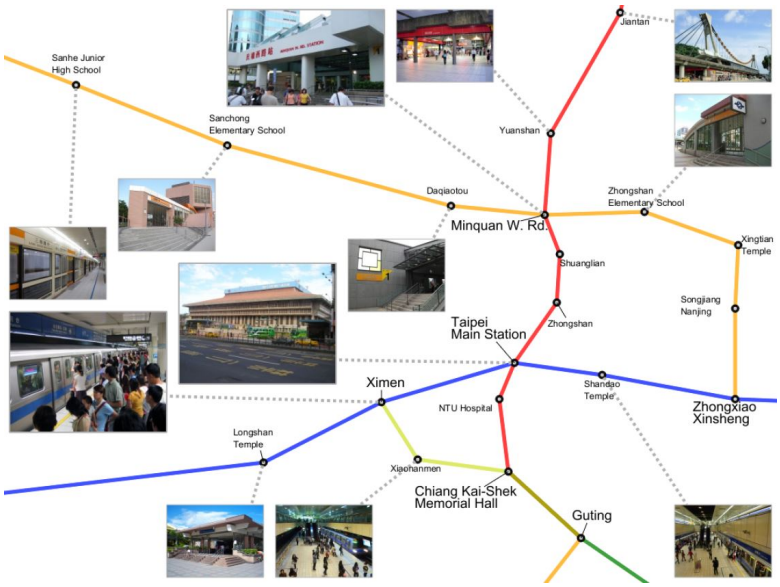


Figure 2.10: Result obtained by Wu et al. Source [35].

2.3 Mixed labeling

Götzelmann et al. [16] proposed an algorithm for labeling interactive 3D models with both internal and external labels. In their approach, they first perform a region-based segmentation based on color. On this segmented image, they perform skeletonization, construct skeleton graph, and extract stroke path candidates from it. The best candidate is selected upon evaluation of a set of criteria. Qualities such as steepness, curvature, path length, and surrounding space are evaluated. Thus enabling the selection of the best path segments, where the text is rendered. Results obtained by their approach are depicted in Figure 2.11.

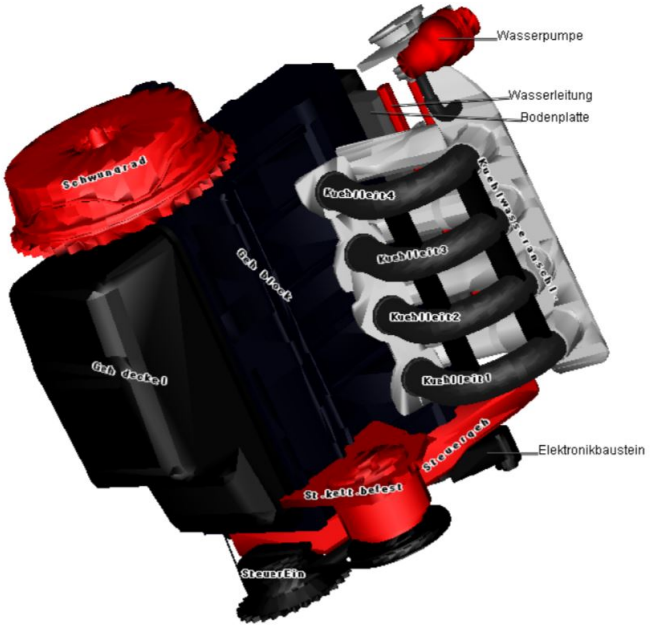


Figure 2.11: Result obtained by Götzelmann et al. Source [16].

Götzelmann et al. proposed further methods for mixed labeling of interactive 3D scenes [15][17]. These methods share the core idea. For a given object, they calculate optimal internal and external label positions. Layout manager chooses which label to place. However, the internal label must fully overlap its object. Thus an internal label might not exist. To add, as they evaluate internal and external labels independently, an overlap of an internal label with a leader line can occur.

2.4 Chapter summary

I described a large number of methods utilized in the field of labeling. I included internal labeling methods in section 2.1, external labeling methods in section 2.2, and mixed labeling methods in section 2.3. Of course, not all existing methods were mentioned in this chapter.

I aim to position the labels both internally and externally, and as such, the most appropriate methods are those of Götzelmann et al. [15][16][17]. However, these methods still have certain limitations. They must place the internal labels entirely inside of the object they label. To add to that, cases, when the leader line overlaps the label, can occur.

I intend to remove these shortcomings by allowing internal label positions only partly overlapping its respective object and not allowing the label to leader line overlaps. Therefore, the method proposed in this work has the potential to contribute to the already existing palette of labeling algorithms.

Chapter 3

Design

In this chapter, I describe the algorithm and the problems that connect to it. Firstly, in section 3.1, I describe the task at hand. In the subsequent section 3.2, I bring an overview of the general techniques, which I will often use, albeit in some altered version, in the following sections. Section 3.3 brings an overview of the algorithm, coupled with an outline of the problems that arise from some particular requirements on the label layout. All subsequent sections focus on describing the solutions to the problems outlined in section 3.3.

3.1 Problem description

This section describes the problem I tackle in this master thesis. In the beginning, there is a scene containing some objects to which I would like to assign labels. It might be a 3D model, a static image or, for example, a map. The particular form of the data depicted in the scene is not too important, as long as there is a way to distinguish between the foreground and the background, i.e., the information carrying data I would like to label and the empty space between them. I require the object segmentation because I need to know which objects are present at a given position in the scene, and I store this information in the *id buffer*.

The task is to assign the labels to the selected or all of the objects in the scene. In this thesis, I consider three possible ways in which I can link a label and an object it is supposed to label. I can place the label internally; this means that it, at least partially, overlaps the object it marks. In this case, only proximity of the label to the object conveys the association, as there is no line connecting the two to aid it.

Another option is to place the label externally, next to the labeled object, and connect it with a leader line to ease the recognition of the link between the two. In this case, the labels are placed on an extruded outline of the objects. This way, I prevent overlaps and assure short leader lines, which is one of the requirements of an external label layout [4]. Furthermore, there is only a negligible waste of free space, i.e., such a position in the screen space where no objects are present, and the external labels are placed with higher versatility.

A situation might arise that the leader line is not needed. For example, an object might be spatially isolated from others present in the scene. Since there are no competing objects for the association by proximity, there is no need for the leader line, and I can remove it. This type of situation might occur even in more densely populated regions of the scene. Lack of competing objects in a given region of the scene is sufficient to discard the use of the leader line. This method of placing the label externally but without the unnecessary leader line constitutes the third way of label placement.

3.2 Techniques in general

There are two techniques, which I frequently utilize in the calculation of the data I require for some step of the algorithm. Although not used in the most general form, but rather with some alterations made to customize them for the particular use case, I employ these techniques in a large number of steps of the algorithm. Furthermore, they ensure sufficient speed of the calculation required for the algorithm to run at interactive frame rates. The following subsections take a closer look at these techniques, which are called jump flooding and scattering. I also include an overview of summed area table, since I encode many textures in this form. I decided to put this section here, so that you, the reader, can familiarize yourself with them before the discussion of the algorithm. This way, the description of the algorithm is more fluent and allows for an accessible understanding.

3.2.1 Jump Flooding

Jump flooding is an algorithmic paradigm suitable for parallel calculations. I can use it when calculating the Voronoi diagram, distance transform, or summed area tables. This paradigm is generally useful when we are faced with a task to spread information contained in any given pixel over the whole grid, in our case, usually, represented as a texture. Some possible usages of this technique were introduced by Rong and Tan [28], who demonstrated its possible usages by calculating Voronoi diagrams.

Let us consider a grid of pixels with arbitrary values. Our target is to propagate the values of all the pixels into every point of this grid. If we would like to solve this problem for just one pixel, a simple way to achieve this would be to flood from this pixel outward. In the first wave, only the neighboring pixels would receive the content; in the second wave, the content would be spread to the pixels neighboring those pixels. Effectively, the information is spread in steps of length one. Rong and Tan [28] deal with this task by propagating the information in rounds with varying step lengths. They either start with large step size and halve it, or a small one and double it. In each step, the information from every pixel is spread in horizontal, vertical, and diagonal directions.

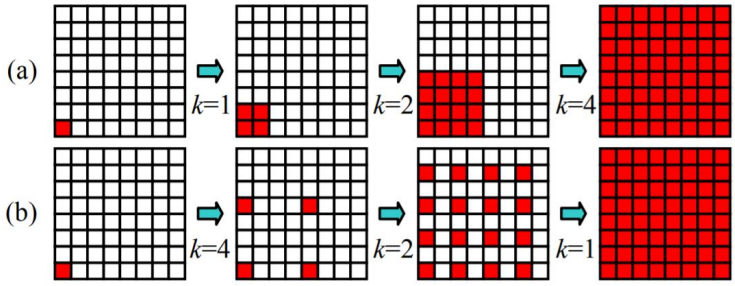


Figure 3.1: Jump flooding of information contained in the red pixel by either doubling the step size (a) or halving it (b). Source [28].

Figure 3.1 shows how this works when propagating a single pixel to all the other pixels in the grid. Jump flooding takes only $\lceil(n)\rceil$ steps on a grid with dimensions of n x n, compared to n steps it would take by the simple method outlined at the beginning of this section. This technique is advantageous and by changing what pixels do with the information and what direction they propagate the information to, various computations can be sped up. If we want to calculate a Voronoi diagram from a set of seeds, we keep the lowest distance from each seed in each pixel. The summed area table can be calculated by propagating the information only to the right and up while adding the value stored in the pixels with the propagated one.

3.2.2 Scattering

In general-purpose computing on graphics processing units (GPGPU) applications, the programmable pipeline of the graphics card is utilized for various kinds of operations. These are generally highly parallel in nature. The fragment processors can perform gather operation, as it corresponds to a simple texture search. However, they are not able to scatter the data, as the output address is fixed to a specific location in the target buffer. Meanwhile, vertex processors can change the location of the current vertex, and therefore are capable of scatter operation, to some degree [20].

Scheuermann and Hensley [31] proposed an approach for efficient histogram generation. They use scattering for this task. They consider an input image, for which they want to compute a histogram, i.e., sort the pixels of the image into buckets based on their brightness. They render one point primitive for every pixel of the input image and perform the bin selection in the vertex shader. A bin is a location in a 1D bin texture. The fragments are rasterized into locations they desire. They configure hardware blend units to add the incoming fragments and thus obtain the histogram.

This approach can be modified for various use cases. I can add the values of the pixels as Scheuermann and Hensley [31] did. Or I can select the maximum value by redirecting all of the pixels of the input image at one location but into different depths and use the depth test to keep only the fragment with the lowest/highest depth. I can also render a line instead of the one pixel

primitive. When the line is rasterized, there are more fragments per pixel in the input image — effectively adding the pixel value to more buckets.

3.2.3 Summed area table

The summed area table, SAT for short, is a data structure and an algorithm utilized for efficiently calculating the sum of values of a rectangular subset of a grid. It was first introduced by Frank Crow [12] in 1984. Input to the algorithm is a rectangular grid. The algorithm outputs a rectangular grid of the identical dimension to the input grid. Every point of the output grid contains the sum of values of the input grid over rectangle determined by the point itself and bottom left corner. Therefore, the summed area table is also known as the integral image. The value $S(x, y)$ of the point with the coordinates (x, y) in the summed area table is:

$$S(x, y) = \sum_{x'=0}^{x} \sum_{y'=0}^{y} i(x', y') \tag{3.1}$$

where $i(x, y)$ is the value of the input grid at coordinates (x, y).

Calculating the sum of the values of the input grid, the black rectangle in Figure 3.2, is an easy operation with the summed area table since the following is true:

$$\sum_{\substack{X_L \leq x \leq X_R \\ aY_B \leq y \leq Y_T}} i(x, y) = S(X_R, Y_T) - S(X_R, Y_B) - S(X_L, Y_T) + S(X_L, Y_B) \tag{3.2}$$

Thus, calculating the sum of a rectangular subset of a grid requires only four lookups to the summed area table.

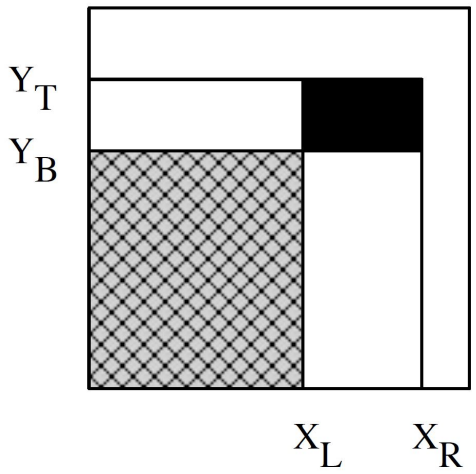


Figure 3.2: Summed aread table. Source [19]

Summed area table calculation

From the definition of the summed area table, it is evident that its calculation is not a difficult task. However, for purposes of this thesis, the implementation must be fast enough to allow for real-time performance of the proposed method. Therefore a GPU implementation is required. Hensley et al. [19] proposed a method to calculate the summed area table in $O(\lceil log(n) \rceil)$ time, where n is the maximum from the width and the high of the grid. They utilized a technique known as recursive doubling.

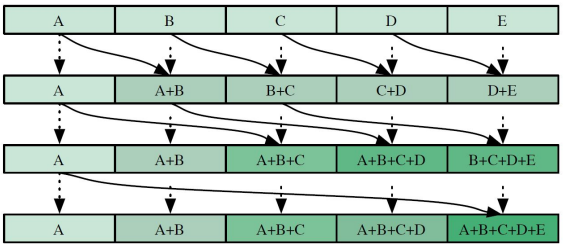


Figure 3.3: Recursive doubling in 1D. Source [19]

Figure 3.3 depicts the progress of calculation in a 1D problem. In the first step, the value from the pixel left of it is added to every pixel. In the second step, the step size doubles, and the value of the pixel located two grid cells to the left is added to every pixel. Step size is doubled with every iteration. For a 2D problem, the algorithm functions in a vertical and a horizontal phase. First, it calculates row sums in the horizontal phase. Subsequently, values obtained in the previous phase are summed column-wise. Implementation usually utilizes two textures. One texture to read from and one to write to. These textures are swapped after every iteration.

3.3 Overview of the method

I will first define the necessary terminology for the task at hand. Given a model M comprising n parts $O_i, i \in \{1, ..., n\}$ located in a scene, I consider a projection of the models M into the screen space $\mathbb{S}$ and denote it as $\mathbb{A}$. Furthermore, I denote the projection of part O_i as A_i, an area where the projection of part O_i is present. M can be a 3D model or a 2D image. I only require the projection of M to be known, as it is stored in the *id buffer* and utilized for the calculations. I will refer to the part O_i of the model also as the object O_i. For semi-transparent objects, I use the approach of Čmolík and Bittner [11], who discard portions of the A_i, where the object is too transparent or where it is occluded by another opaque object.

Furthermore, I require a short textual annotation t_i for each object O_i, a label, which should be placed in the scene to aid recognition of a given part. I assume the dimension d_i of t_i to be known prior to the calculations of the algorithm ($d_i = (w_i, h_i)$ is a vector of the label width and the label height). I

call the projection of the boundary of the extruded model M into the screen space the extruded silhouette.

I denote l_i as a label candidate for area A_i. I denote the label box (imaginary rectangle encompassing the label) corresponding to the label candidate l_i as L_i. Dimension of the label box L_i is the dimension d_i of the label text t_i. Label candidate can be internal or external and I denote them l_{Ii} and l_{Ei}, respectively. However I will use only designation l_i, as it is apparent whether I in fact mean l_{Ii} or l_{Ei}.

Label candidate l_i is a pixel in $\mathbb{S}$. To every label candidate l_i corresponds a unique position of L_i in $\mathbb{S}$. In the following text I use A_i to describe both the projection of the object O_i as well as the external label candidates, as the two set are identical.

The algorithm can be roughly described by the pseudo-code in Algorithm 2.

```
Result: Positions of the labels for the given scene
For each area A_i, i ∈ {1, ..., n} establish label candidates;
Evaluate the fitness of each label candidate according to the labeling
 criteria;
while There is an unlabeled area left in 𝔸 do
    Select unlabeled area A_i with the lowest capacity C(A_i);
    Discard the unsuitable candidates based on overlaps;
    Find the best internal label candidate for A_i;
    if fitness of the best internal label candidate lower than user
     threshold t then
        Find the best externall label candidate for A_i;
        if external label candidate exists then
            Discard the internal candidate;
        else
            Keep the internal candidate based on user preference;
        end
    end
    if label was placed internally then
        Discard external label candidates for A_j, i ≠ j where
         candidate leader line would interset the placed label;
    else
        Discard internal label candidates of A_j, i ≠ j where the
         candidate would interset the leaderline of the placed label;
    end
end
```

Algorithm 1: Algorithm overview

Firstly, I need to determine the internal and external label candidates. For O_i, I consider every pixel of A_i to be a candidate for an anchor of the external label, and also a label candidate l_i. I am trying to position the external labels in a way so that they are touching the extruded silhouette with one of their corners. Since there is precisely one leader line l_{l_i} connecting an anchor point with the closest point of the extruded silhouette, depicted in Figure 3.4b,

there is exactly one leader line associated with each anchor point. Therefore, the task of finding l_{l_i} is the same as finding an anchor to which the l_{l_i} is linked. I assume the position of the label box L_i to be dictated by the direction of l_{l_i}.

The situation for internal labels is similar. I consider label t_i of O_i internal if it overlaps A_i at least partially ($L_i \cap A_i \neq \emptyset$) and is associated by proximity and not by a leader line. It is sufficient to choose one particular point of the label to represent the label candidate since dimension d_i is known. I selected bottom left corner of the label to be this point. To calculate all of the internal label candidate for object O_i, I need to extend area A_i down by the and left by the dimension d_i. This yields area AI_i, each pixel of which represent an internal label candidate. This area is depicted in Figure 3.4a.

After the label candidates are known, I need to be able to evaluate their fitness. For this purpose, I impose criteria on them. Criteria differ based on the type of candidate. However, they are mainly based on the evaluation of the ambiguity of the label candidate with respect to the labeled object.

The external label candidates are subjected to the following criteria:

- **Leader line length** – leader line should be as short as possible
- **Anchor salience** – the anchor should be a salient point of A_i
- **External label salience** – the label should be placed in a position, where it can be associated with A_i with as little ambiguity as possible.
- **External label overlap** - the label does not overlap area $\mathbb{A}$, nor does it overlap other labels.

Meanwhile, the internal labels are evaluated based on the following criteria:

- **Internal label salience** - the label box L_i should overlap the object A_i as much as possible, and at the same time the overlap with $A_j \neq A_i$ should be minimal for all $j \in \{1, 2, 3, ..., n\} \setminus i$. Further, a portion of the label candidate that is external ($L_i \cap (\mathbb{S} \setminus (A))$) should be associated with A_i with as little ambiguity as possible.
- **Internal label overlap** - the label does not overlap other labels.

After I evaluated all the label candidates, the main loop of the algorithm can run and iteratively place labels in the scene. First, an area to be labeled must be selected. For this purpose, I take a weighted sum of the internal label candidates, called a capacity $C(A_j)$, for each area A_j. I select the area A_i with the smallest capacity $C(A_i)$ from yet unlabeled areas. Subsequently, I try to place the internal label. If it is not possible to place it, or the fitness of the selected candidate is lower than a user set threshold t, I attempt to place an external label. If the external label cannot be placed, based on the user preference, the internal label can be placed instead, if it exists. The main loop of the algorithm runs until there are no unlabeled areas left.

3.4 Label candidates

The first step of the algorithm is to determine the label candidates. The algorithm is GPU evaluated. Therefore it is suitable to represent a candidate as a pixel in the screen space. I also want to eliminate transfers between CPU and GPU. Representing candidates as pixels in some buffer in the GPU memory does accomplish that. However, the particular form of the candidate representation differs based on whether I deal with an internal candidate for a label or an external one.

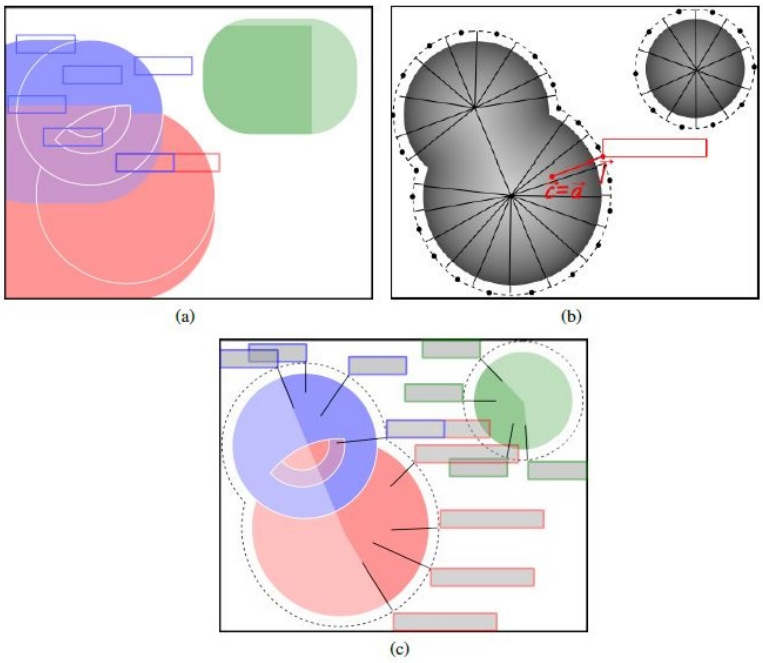


Figure 3.4: The internal label candidates (a), the internal Voronoi diagram of the extreduded silhouette (b), the external label candidates (c).

3.4.1 External label candidates

I represent the external label candidate by its anchor. Each anchor has exactly one closest point on the outline of the model. This is depicted in Figure 3.4b, where the anchor a has a closes point l on the outline. The position of the label box is dictated by the two, with a corner of the label box being the point l. Therefore, a leader line, which is determined by an anchor and a point on the outline, can be viewed as being fixed to its anchor. Moreover, by fixing the position of the label box to the direction of the leader line, I can achieve it dictated by the position of the anchor. Consequently, one external label candidate corresponds to one anchor. Based on the angle between the leader line from point a to point l and the horizontal axes, a different corner of the label box touches the leader line. For the angle in the interval $(0°, 90°]$ it is the bottom left corner, in the interval $(90°, 180°]$ it is the bottom right corner, in the interval $(180°, 270°]$ it is the top right

corner, and in (270°, 360°] it is the top left corner. Figure 3.4 (c) depicts these different label orientations. For each object O_i, its anchors are all of the points of A_i, except for the ones discarded due to occlusion by other objects, or abundance of transparency, utilizing the method of Čmolík and Bittner [11], as was mentioned in the algorithm description section.

3.4.2 Internal label candidates

Internal label candidates are somewhat more difficult to obtain. Whereas in case of external label candidates for O_i, it was simply all of the points in the area A_i, in case of the internal label candidates, it will have to be a superset of A_i. Since any label that would even partially overlap the area A_i is considered a label candidate for O_i, I need to reflect this in the set of internal label candidates. I define the bottom left corner of the label candidate to be the identifying point of the whole candidate, as the position of the bottom left corner coupled with the known dimensions of the label sets a unique label position. Label dimension for area A_i are $d_i = (w_i, h_i)$, where d_i is a vector of the label dimensions, w_i is the label width and h_i is the label height. By extruding the area A_i by the w_i left and the h_i down, I obtain the set of internal label candidates for O_i - AI_i. This idea is depicted in Figure 3.4c.

I can calculate all the $AI_i, i \in \{1, 2, ..., n\}$ efficiently and simultaneously by a modifying the jump flooding Algorithm. In general case of the jump flooding, every pixel of the grid, in this case of the screen, has an access to the value of every other pixel of the grid at some point during the algorithm run. However, it is not necessary in this case, as only the values of the pixels incoming from the right and up, which are closer than w_i or h_i, respectively, are relevant for any given pixel. Consequently, I can obtain AI_i from A_i by propagating the information of all the pixels in the screen space $\mathbb{S}$ only a limited distance to the left a down.

I can divide the propagation into two independent phases by first propagating the pixel values to the left and then down. For either one of the directions, the following holds true. By halving the step size from initial $s_0 = 2^K, K \in \mathbb{N}$ down to $s_K = 1$, where s_k is step size in iteration k of the jump flooding algorithm, the total distance the value from any pixel can be propagated to is:

$$d_p(k) = \sum_{j=0}^{K} 2^j = 2^{K+1} - 1 \tag{3.3}$$

By adding the one pixel that the value originated from, it is apparent that it has now been distributed to 2^{K+1} pixels in total. Therefore, it would not be hard to obtain AI_i from A_i if the dimension of the label was $d_i = (w_i = 2^{k_{i0}}, h_i = 2^{k_{i1}}), k_{i0}, k_{i1} \in \mathbb{N}$. However, this is not always the case. Fortunately, I am equipped with building blocks of size $d_p(\mathbb{N}) = \{1, 3, 7, 15, ...\}$, which I can combine to obtain any number $x \in \mathbb{N}$. Furthermore, it is apparent that decomposition

$$x = \sum_{l=0}^{p} d_p(l) = \sum_{l=0}^{p} \sum_{j=0}^{K_l} 2^j \tag{3.4}$$

requires at most $log(x) + 1$ terms. This, in turn, yields an upper bound on the time complexity, as the number of passes, horizontal and vertical, in the jump flooding algorithm is strictly constrained by:

$$O(log(w_i) * (log(w_i) + 1) + log(h_i) * (log(h_i) + 1)) \tag{3.5}$$

This is much faster than a naive approach that would propagate a value by the distance of 1 in each iteration. Or a distance transform utilizing weighted Chebyshev metric, which would run in $log(n_s)$ steps, n_s being the number of pixels of the screen row, but would have to be computed for every label separately, as every label would require different weights for the metric. It is also not possible to store the origin of the propagated pixel and perform just one complete loop with initial step size sufficient to cover the whole screen and stop the propagation of those pixels that would get out of bound, given by the dimensions of the label. The problem is that for each id, there might be a different origin for the pixel.

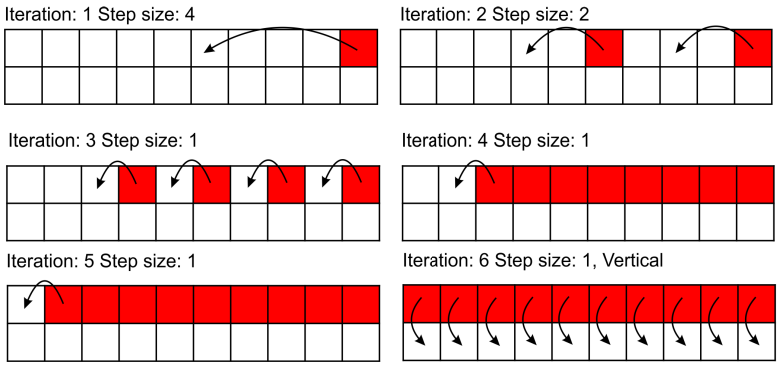


Figure 3.5: Propagating an id on a 10x2 grid using variation of jump flooding

Figure 3.5 shows the propagation of one seed on a 10x2 grid. Because the dimension of the grid is quite small, the number of steps is above expectation, as iterations with small step size are repeated. Also, the width is 10 pixels; this means that there are 9 extra pixels where the seed needs to be propagated. Decomposition of 9 into the new base i $9 = 7+1+1$, or $9 = 4+2+1+1+1$ in terms of step sizes. Distribution of pixel value in both horizontal and vertical direction is also possible but will not result in a significant speed up.

Candidates for every object O_i can be calculated simultaneously by introducing propagation mask. It directs which ids should be propagated in a given step. Step sizes are chosen so that every id is propagated exactly the right distance.

3.5 Modelling the criteria

Sets of pixels representing both the internal and external label candidates are now calculated for every object O_i. The external label candidates for area O_i are equal to its projection, the area A_i. The internal label candidates AI_i are computed from A_i by jump flooding. In order to select one of these candidates for every object, Fitness function must be introduced. This section list the criteria utilized for the selection of label candidates. While there are some differences between the criteria for internal and external labels, they are based on similar requirements. The labels should be attributable to their objects with minimal ambiguity. Since this thesis strives to label without leader lines, if possible, ambiguity plays a significant role. Each criterion is modeled as a fuzzy set with a membership function:

$$C_i : \mathbb{S} \to [0, 1] \tag{3.6}$$

The value of $C_i(l_j)$ describes how well the label candidate follows a given criterion. If the value is high, the candidate is suitable, if it is low, the candidate is unsuitable from the point of view of the criterion. For the aggregation of individual criteria into fitness function F, I utilize Multiple Criteria Decision Making based on fuzzy logic [6]. Criteria are aggregated in a non-compensating way. This means that strongly meeting one criterion does not compensate for not meeting another. In order to modify the strength of each criterion, I utilize weights. Criteria are aggregated with Natural Fuzzy Conjunction (Natural T-Norm), which corresponds to the standard multiplication. Fitness of the label candidate is given by the fitness function F, which is defined as follows:

$$F(l) = \bigwedge_{i=1}^{6} C_i(l)^{w_i} = \prod_{i=1}^{6} C_i(l)^{w_i} \tag{3.7}$$

where w_i is the weight of criterion C_i. The criteria utilized to determine the fitness of internal and external label candidates are not identical. Furthermore, for all of the pixels, which are not label candidates the value of all criteria is equal to zero.

3.5.1 Label salience

The salience of the label as a whole is a pivotal criterion in evaluating label ambiguity. When leader lines are not utilized to link a label with a corresponding part, the salience of the label candidate is the only measure of how ambiguous the position of the label for a given object A_i would be. I distinguish two types of salience - internal and external. Internal label salience aims to describe the level of ambiguity in the placement of internal label, while external label salience describes ambiguity in the placement of the external label. Since I consider a label to be internal even if it only partially overlaps $\mathbb{A}$, external label salience criterion is applicable to it as well.

External label salience

Figure 3.6 depicts possible problems with the ambiguity of label placement. If the labels are placed in a manner shown in Figure 3.6a, it is impossible to correctly assign them to the labeled objects. When label positions shift slightly, identification of which label belongs to which object becomes much easier, as depicted in Figure 3.6b.

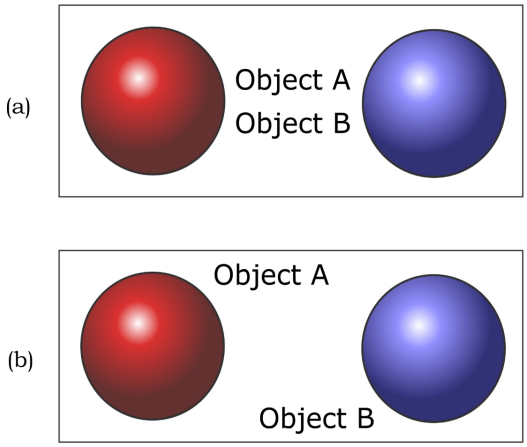


Figure 3.6: Possible problems with ambiguity

The solution to why this is the case is depicted in Figure 3.7. The Voronoi diagram is constructed here with the individual regions having the appropriate color tones. Therefore, I am lead to conclude that the labels should have a maximal overlap with the region of object they label, all the while having a minimal overlap with the regions of objects, they do not label.

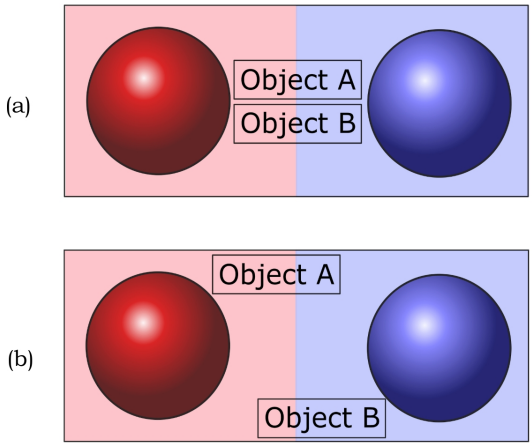


Figure 3.7: Possible problems with ambiguity solution

Prior to the solution, I provide an overview of the terms utilized in the following text. I define the outline P_i of the area A_i as $P_i = \partial A_i$, where ∂A_i represents a boundary of A_i. The outlines of all areas serve as seeds to

calculate the Voronoi diagram of the scene. Region R_i, meaning points $x \in \mathbb{S}$ closer to P_i than to any other $P_j, j \neq i$ is defined as follows:

$$R_i = \{x \in \mathbb{S} | (\forall j \neq i)(d(x, P_i) \leq d(x, P_j))\} \tag{3.8}$$

where d is a distance function, in this case euclidean distance, and $d(x, P_i)$ (i.e. the distance of a point from a set) is defined as:

$$d(x, P_i) = min\{d(x, y) | y \in P_i\} \tag{3.9}$$

There are two requirements that a label should meet, as mentioned in the preceding text. At first, I will model them separately, so there is no confusion as to what they describe. The first requirement states that the label should overlap the region of the objects it marks as much as possible. Following criterion models this requirement:

$$C_{1a}(L_i) = \frac{S(R_i \cap L_i)}{S(L_i)} \tag{3.10}$$

where L_i is a label box of a label candidate for area A_i and $S()$ is a surface function. Meaning that $S(L_i)$ is the area of the label box and $S(R_i \cap L_i)$ is area of the label box located in region R_i.

The second requirement states that the label should overlap regions $R_j, j \neq i$ as little as possible. I can model this criterion as a lack of overlap. By not overlapping the region R_j at all, the label candidate will meet the criterion perfectly. Based on this, the criterion for R_j reads:

$$C_{1b_j}(L_i) = 1 - \frac{S(R_j \cap L_i)}{S(L_i)} \tag{3.11}$$

However, there is not only one other region a label could overlap, but there are more of them. I can utilize fuzzy conjunction to aggregate the requirements for all the $R_j, j \neq i$ as follows:

$$C_{1b}(L_i) = \prod_{j=1, j \neq i}^{n} (1 - \frac{S(R_j \cap L_i)}{S(L_i)}) \tag{3.12}$$

To obtain the criterion C_1, I aggregate the partial requirements modelled by C_{1a} and C_{1b}. The result reads as follows:

$$C_1(L_i) = \frac{S(R_i \cap L_i)}{S(L_i)} \times \prod_{j=1, j \neq i}^{n} (1 - \frac{S(R_j \cap L_i)}{S(L_i)}) \tag{3.13}$$

Voronoi diagram of the scene $\mathbb{S}$ can be obtained in two simple steps. First, I need to calculate the outlines $\forall P_i, i \in \{1, 2, 3, ..., n\}$. This can be achieved by detecting the discontinuities in the *id buffer*. Figure 3.8a depicts an illustration of the *id buffer*, while Figure 3.8b shows the discontinuities in the buffer - the outline.

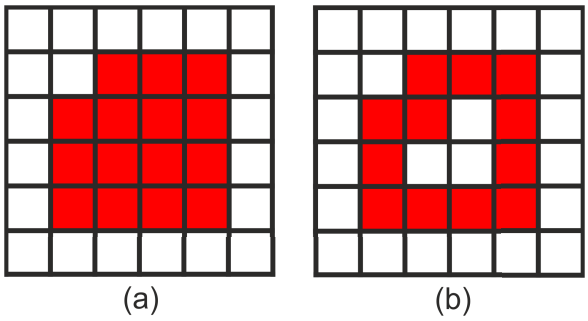


Figure 3.8: Example *id buffer* (a) and the outline (b) calculated from it

The outline serves as seeds for calculation of the Voronoi diagram. I utilize the method of Rong and Tan [28] for this task. Calculation of the Voronoi diagram, or at least its faithful approximation, is a fast operation. However, determining the surface overlap of the label box L_i with any given region R_j poses problems. Accessing all of the pixels that comprise the label box in a fragment is an option but would result in poor performance. The Voronoi diagram cannot be encoded as summed area table, because each bit of each pixel is reserved for an object with a given id, and summed area table calculation would result in overflows and therefore would not be possible.

One possible solution I came up with is to create a tile texture, in which each tile holds a mask containing a unique region R_j. Pixel value in the $i-th$ tile at position (x, y) is:

$$t^i_{xy} = \begin{cases} 1 & \text{if } (x,y) \in R_i \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

The value t^i_{xy} communicates information whether region R_i is present at a given position in $\mathbb{S}$. Indexation of the tiles in the tile texture is not particularly important, as long as it is reproducible. The tile texture can be encoded as a summed area table. This, in turn, allows for efficient calculation of $S(R_j \cap L_i)$, because only four texture lookups are needed.

However, there is no way to know which tiles should be accessed. Consequently, all the tiles need to be checked. I can overcome this problem by an approach similar to that of in section 3.4.2. I can take the Voronoi diagram as an input. From it, I compute the tile ids that need to be accessed for every candidate. I save them to a new buffer.I calculate it by propagating each id left and down by the width of the widest label and height of the highest label, respectively. When deciding which tiles to access I can read the value from this buffer first. Meaning of the value is: if it contains an id i, then the label probably overlaps region R_i. It is not a certainty that the overlap is present because I used the largest dimensions of the label.

Internal label salience

There are significant problems that might occur when placing the label internally. It is associated with the object it marks only by proximity.

Position of the label over the object cannot be arbitrary, as some positions present ambiguous states.

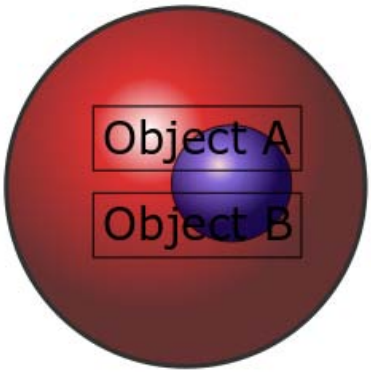


Figure 3.9: Possible problem with the ambiguity of internal label placement

Figure 3.9 shows a possible issue with ambiguity for internally placed labels. This sort of labeling is not acceptable, because it fails to communicate the label object association. Criterion similar to anchor salience proposed by Ali et al. [4] regarding anchor salience is needed to solve this problem. However, if only one point of the label candidate would be used to decide the optimal position, the resulting label positions will be suboptimal. Furthermore, the placement of the labels might be ambiguous. As Figure 3.10 shows, only positioning the center of the label boxes over the most salient point of the respective areas produces suboptimal results.

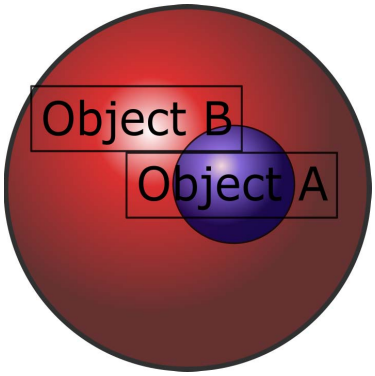


Figure 3.10: Possible problem with ambiguity of internal label placement

I must introduce a more intricate method of evaluating label salience. External salience of a label, as described in section 3.5.1 is based on assessing the ratios of the label present in different regions. Internal salience of a label is more complex. Internal anchor salience is the distance from the nearest boundary of some area ∂A_j. For salience to have the shape of a membership function, I normalize it by the largest salience. Also, as I am interested in the salience of internal areas, I define the salience of points in $\mathbb{S} \setminus \mathbb{A}$ to be 0.

More formally by normalized salience $s(x)$ of the point x, I understand the following:

$$s(x) = \begin{cases} \frac{\min\{d(x,y)|y \in \bigcup_{i=1}^{n} \partial A_i\}}{\max_{z \in \mathbb{A}} \min\{d(z,y)|y \in \bigcup_{i=1}^{n} \partial A_i\}} & \text{if } x \in \mathbb{A} \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

where $d(x, y)$ is a distance function. In this case, Euclidean distance. Whereas the anchor salience describes a property of a single point - an anchor, I need to describe salience of an area - a label box. Point salience does not sufficiently describe the whole label. Any point of the label box area does not convey representative information about the partitioning of the label into distinct areas A_j.

For this reason, I need to aggregate information about the salience of all the points comprising the label box, into one value. I do this by taking the mean salience over the whole label box. However, the label box might be positioned over multiple areas A_j. If I am attempting to label area A_i, I need to maximize the mean salience over this area. The same is not true for areas $A_j, j \neq i$, as any overlap with them is undesirable. Figure 3.11 depicts an optimal solution, where labels are placed over salient areas of their respective objects.

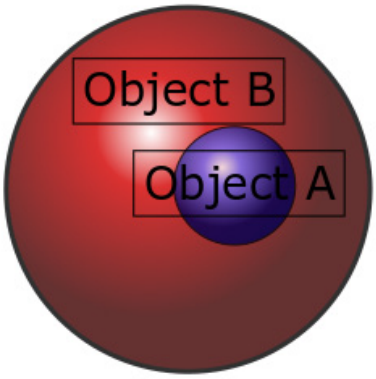


Figure 3.11: Optimal internal label placement

Similarly to section 3.5.1, the internal salience criterion needs to reflect two requirements. The labels should be placed over the objects they label, and the labels should not be placed over the objects they do not label. Form of the criteria is therefore identical to that of C_1 and it reads:

$$C_2(L_i) = S_a(L_i, i) \times \prod_{j=1, j \neq i}^{n} (1 - S_a(L_i, j)) \tag{3.16}$$

where $S_a(L_i, j)$ is the mean salience of the candidate label box L_i for area A_i with respect to area A_j and $S_a(L_i, j)$ is calculated as follows:

$$S_a(L_i, j) = \frac{1}{|L_i|} \times \sum_{x \in L_i \subset \mathbb{S}} (s(x)) \tag{3.17}$$

where $|L_i|$ is the number of pixels the label box comprises of.

I split the criterion C_2 into two parts. The criterion C_{2a} allows controlling the influence that the salience of the label with respect to the object it labels. It models the first requirement of the two.

$$C_{2a}(L_i) = S_a(L_i, i) \tag{3.18}$$

The criterion C_2b can be used to prevent any overlaps with foreign objects. It models the second requirement of the two.

$$C_{2b}(L_i) = \prod_{j=1, j \neq i}^{n} (1 - S_a(L_i, j)) \tag{3.19}$$

Salience for every point $x \in \mathbb{S}$ can be efficiently calculated along with the calculation of Voronoi diagram by jump flooding, as shown by Rong and Tan [28]. As in section 3.5.1, there arises a problem how to efficiently calculate the sum $\sum_{x \in L_i \subset \mathbb{S}} (s(x))$. It is possible to encode the buffer containing salience of points as summed area table. However, when retrieving the information back, I cannot decompose the sum over the label box into partial sums over A_i. The solution is the same as in section 3.5.1. I distribute the information about point salience into individual tiles, based on id, each of which corresponds to a unique object. Furthermore, I encode the tile texture as a summed area table, which enables me to perform lookups efficiently for every id. To reduce the number of tiles into which a lookup needs to be performed, I utilize the precomputed id structure from section 3.5.1.

3.5.2 Anchor salience

External labels placed with leader lines utilize criteria similar to those of Ali et al. [4] and many others. Anchor salience is one of the criteria imposed on anchors. It forces the selection of a label candidate with an anchor that is clearly attributable to the labeled object. The problem with anchor salience was already discussed in section 3.5.1. Anchor salience is the distance to the nearest point from the $\bigcup_{i=1}^{n} \partial A_i$ normalized by the largest salience - d_{max}. The criterion on anchor salience thus reads:

$$C_3(l_i) = \frac{d(a, \bigcup_{i=1}^{n} \partial A_i)}{d_{max}} = s(a) \tag{3.20}$$

where a is an anchor corresponding to a label candidate l_i.

3.5.3 Leader line length

According to Ali et al. [4] a label should be close to the corresponding object. Čmolík and Bittner [10] utilize the same method and require the leader lines

to be as short as possible. Leader line length can be easily calculated by distance transform of the extruded silhouette of area $\mathbb{A}$ serving as seeds, as accomplished by Rong and Tan [28].

Furthermore, the distance function utilized to determine the distance of the anchor from the boundary can vary, based on the required label layout. It can be Euclidean distance or some modification of Chebyshev distance for layout with leader lines oriented vertically or horizontally. Choice of the metric in turn affects leader lines themselves. Different points on the silhouette can be closest to a given anchor based on the metric. As with anchor salience, the length of the leader lines must be normalized for it to serve as a membership function. The criterion for leader lines reads:

$$C_4(L_i) = 1 - \frac{|a - e|}{d_{max}} \tag{3.21}$$

where a is an anchor corresponding to the label candidate l_i and e is the closest point on the silhouette. Vector $l = e - a$ is the leader line corresponding to the label candidate l_i.

3.5.4 Label to model overlap

External labels should not overlap any of the objects. I can accomplish this by creating an occlusion mask of the scene. Any pixel of the occlusion mask can have two values - one or zero based on the following rule:

$$p(x, y) = \begin{cases} 1 & \text{if } (x, y) \in \mathbb{A} \\ 0 & \text{if } (x, y) \in \mathbb{S} \setminus \mathbb{A} \end{cases} \tag{3.22}$$

where $p(x, y)$ is the value of the pixel in occlusion mask at position (x, y). Criterion modeling overlap of the external label with the model is as follows:

$$C_5(L_i) = \begin{cases} 1 & \text{if } \sum_{(x,y)\in L_i} p(x, y) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.23}$$

By encoding the occlusion mask a summed area table, I can achieve significant speed up. Only four lookups are needed to determine whether a label overlaps the model - area $\mathbb{A}$, i.e., the sum in C_5.

3.5.5 Label to label overlap

Label to label overlap can be eliminated by introducing an occlusion mask for the labels. The labels occlusion mask must be kept separate from the scene occlusion mask. This is because both the internal and external labels cannot overlap other labels but only the external labels cannot overlap the model. Meanwhile, the internal labels must do so. The occlusion mask for the labels can be initiated with zeros for every pixel $p(x, y) = 0, \forall (x, y) \in \mathbb{S}$ and updated to one if a label is placed over the position. The value $p(x, y)$ of the pixel on position (x, y) is therefore:

$$p(x,y) = \begin{cases} 1 & \text{if } \exists L_{Pi}, (x,y) \in L_{Pi} \\ 0 & \text{otherwise} \end{cases} \tag{3.24}$$

where L_{Pi} is already placed label for area A_i. The criterion for label-to-label overlap therefore reads:

$$C_6(L_i) = \begin{cases} 1 & \text{if } \sum_{(x,y)\in L_i} p(x,y) = 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.25}$$

By encoding the occlusion mask for the labels as a summed area table, look ups can be sped up significantly. Furthermore, as shown by Pavlovec [26], there is a simple way to update the summed area table, as long as the update is restricted to a rectangular area. This method requires only one pass through the shader program.

3.6 Selection of area for labeling

At this point, the label candidates have been established, see section 3.4, and there is a system in place to evaluate the fitness of individual internal and external label candidates, as discussed in section 3.5.

The order in which the labels are positioned over the scene is important because my approach is based on a greedy optimization, and it cannot recover from a bad partial solution. Furthermore, when a label is positioned over the scene, I must discard some of the label candidates. If the label is placed externally, then internal label candidates that would overlap the leader line of the placed label must be discarded. If it is placed internally, then external label candidates with a leader line overlapping the internal label must be discarded. The number of candidates for each label is changed after each iteration when a label is placed into the scene. I will discuss this in more detail in section 3.9.

In order to select an unlabeled area A_i for which the label should be placed next, I calculate the capacity of all the areas and select the unlabeled area with the lowest capacity. By the capacity of an area, I understand the sum over fitness function applied to label candidates.

$$C(A_i) = \sum_{l_i \in AI_i} F(l_i) \tag{3.26}$$

where $F(l_i)$ is the value of the fitness function on label candidate l_i and AI_i is the set of internal label candidates for A_i. However, since I calculate the sum over the internal label candidates, effectively only C_2 and C_6 is applied and $C(A_i)$ is simplified to:

$$C(A_i) = \sum_{l_i \in AI_i} C_2^{w_2}(l_i) \times C_6^{w_6}(l_i) \tag{3.27}$$

where w_2 is the weight of the criterion C_2. Area for which the label should be placed next is selected as:

$$A_{next} = \underset{A_i \subset \mathbb{A},\ A_i \text{ not labeled}}{\arg\min} C(A_i) \tag{3.28}$$

As discussed above, after a label for A_{next} is placed in the scene, capacities of areas A_i change. Therefore they need to be recalculated. It is possible to efficiently add up all of the weighted values of candidates for given area A_i, and thus obtain the capacity $C(A_i)$ by scattering as discussed in 3.2.2. I send the fitness values of the label candidates into bins corresponding to the areas and add them up.

3.7 Finding the best candidate

After the area A_{next}, which is to be labeled next, is selected, the best label candidate of this area must be chosen. At first, only internal label candidates are considered. To choose the best one of them, the fitness function F must be evaluated for every one of them. Although, for internal labels, effectively only criteria C_1, C_2 and C_6 are evaluated.

The best internal label candidate for a given area A_i, in this case A_{next}, is therefore calculated as:

$$l_i^{optI} = \underset{l_i \in AI_i}{\arg\max} F(l_i) \tag{3.29}$$

where AI_i is the set of internal label candidates for area A_i. If the fitness of the best label candidate $F(l_i^{optI})$ is above the user defined threshold t, I place the label of the label candidate l_i^{optI} in the scene. If the fitness is below the threshold t, then I search for the best external label candidate, which is:

$$l_i^{optE} = \underset{l_i \in A_i}{\arg\max} F(l_i) \tag{3.30}$$

In this case, all of the criteria in the fitness function F must be evaluated except for C_2, which is only for internal labels. If the external label candidate l_i^{optE} exists, I place it in the scene.

Label candidates l_i are represented by pixels in a buffer. Therefore the pixel with the highest value can be found utilizing scattering, see section 3.2.2, by sending all of the candidates l_i at one position in a buffer with a depth equal to the fitness of the candidate $F(l_i)$. Taking the one with the highest depth value, I find l_i^{optI} or l_i^{optE}, respectively.

3.8 Overlap elimination

When a label is placed in the scene, it will inevitably interfere with some of the label candidates. The label-to-label overlap is taken care of by courtesy of criterion C_6. However, some problems might still occur. If the last label was placed externally, its leader line could eliminate some of the internal label candidates for other areas. Likewise, if the label was placed internally,

it might eliminate some of the external label candidates with leader lines crossing the internal label.

The label box can be represented as four line segments and a leader line is just another line segment. Furthermore, the line segments comprising the label box boundary are parallel with horizontal or vertical axes, respectively. Consequently, the computation of the intersection between the leader line and the label box is not complicated. Therefore, I can eliminate unsuitable internal and external label candidates easily.

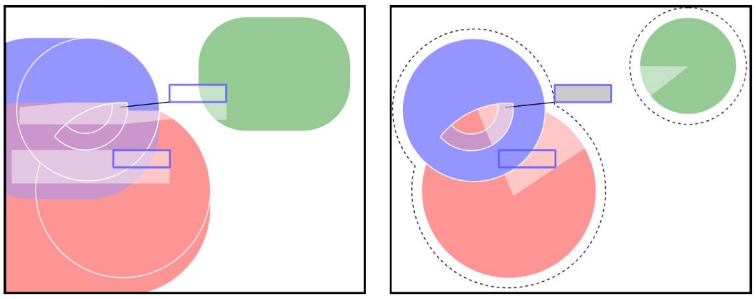


Figure 3.12: Discarded internal and external label candidates, shown in light colour, after an internal (a) and an external label (b) was placed.

Figure 3.12a shows discarded internal label candidates, while Figure 3.12b shows discarded external label candidates. This includes those candidates discarded on bases of criterion C_6.

3.9 Chapter summary

At first, section 3.1 takes a closer look at the problem I am solving in this thesis. Subsequently, in section 3.2 I discuss the general algorithmic approaches, which I utilize later on. In the followings section 3.3, I bring an overview of the proposed method and illustrate the algorithm by the pseudocode in the Algorithm 1. Subsequent chapters discuss individual steps of the algorithm in more detail. At first, I establish both internal and external label candidates in section 3.4. These candidates are represented by single pixels in their respective buffers.

I go on and in section 3.5 I model the criteria with which I evaluate the fitness of individual labels candidates. I pay special attention to criteria that regard label salience, see section 3.5.1, as the method for evaluation of the label box salience, is one of the main contributions of this thesis.

In the following section 3.6 I present the method of determining the order in which the objects should be labeled. Order in which the labels are placed is important because the algorithm utilizes a hungry optimization approach and cannot recover from a bad partial solution. In the subsequent section 3.7, I describe a method to find the best label candidate for the area A_i with the lowest candidate sum, calculated in the previous step. The label candidate l_i

for area A_i with the highest values of the fitness function $F(l_i)$ is placed in the scene. Whether it is placed internally or externally depends on the value of the fitness function and existence of an external label candidate for A_i.

Some label candidates for the remaining unlabeled areas must be discarded after the label l_i is placed in the scene. Consequently, in section 3.8, I present a mechanism of how to discard label candidates that would cause undesirable label - label or label - leader line overlaps.

Chapter 4

Implementation

In this chapter, I will describe the algorithm as designed in chapter 3. Implementation copies the design very well. Therefore, I will limit the discussion almost exclusively to the issues pertaining to the implementation details.

4.1 Implementation technologies

This section describes the technologies that I will utilize for the implementation. The main library is the Tiger library, which requires all of the three other technologies mentioned to function properly. I implemented the actual code with the following versions:

- **OpenGL**: 3.1
- **JOGL**: 2.2.2
- **GLSL**: 3.30 compatible

4.1.1 OpenGL

OpenGL is cross-platform, cross-language API (application programming interface) for creating 2D and 3D graphics [34]. It is implemented for almost all computer platforms. Apart from the hardware implementations, there are also software ones, which necessarily possess lower computational power. OpenGL is widely used also because of the extensive documentation, which is available at its official websites. The core functionality of OpenGL is rendering to frame buffer. Its run is directed by calling functions and procedures.

4.1.2 JOGL

JOGL is a wrapper library, which enables usage of OpenGL commands in Java. Ti access OpenGL, it utilizes JNI (Java Native Interface) calls. It offers access to standard GL and GLU (OpenGL Utility). However, the GLUT (OpenGL Utility Toolkit) library responsible for window calls is not accessible. There is a logical reason for this, as Java has its own libraries for this purpose, namely AWT, Swing, and SWT. Furthermore, Java provides its NEWT(Native Windowing Toolkit)[1]

4.1.3 GLSL

GLSL is a higher programming language, based on the syntax of C. In fact, GLSL is a set of closely related languages. It is utilized for the creation of shaders for every programmable part of the pipeline, namely for fragment and vertex shaders. Similarly to C, it supports cycles, branching, and user-defined functions. However, recursion is not allowed. Shaders are allowed to access certain parts of states of OpenGL.

4.1.4 Tiger

Tiger is a library designed to make the development of multi-pass rendering effects in OpenGL easier. It accesses OpenGL via wrapper library JOGL [9].

4.2 Algorithm overview

The following section gives an overview of the algorithm 2, as described in section 3.3. To recapitulate, I give the algorithm once again, now using the terminology of chapter 3. The algorithm is GPU evaluated. All the utilized structures are kept in the GPU memory to ensure fast calculations. Since I also store the *id buffer* in the GPU memory, I cannot label more than 128 objects (4x32bit color components). Every object has its own bit in the id.

I already described the function of the algorithm in section 3.3. To avoid repetition, now I will illustrate it with the help of the buffers utilized in the computation.

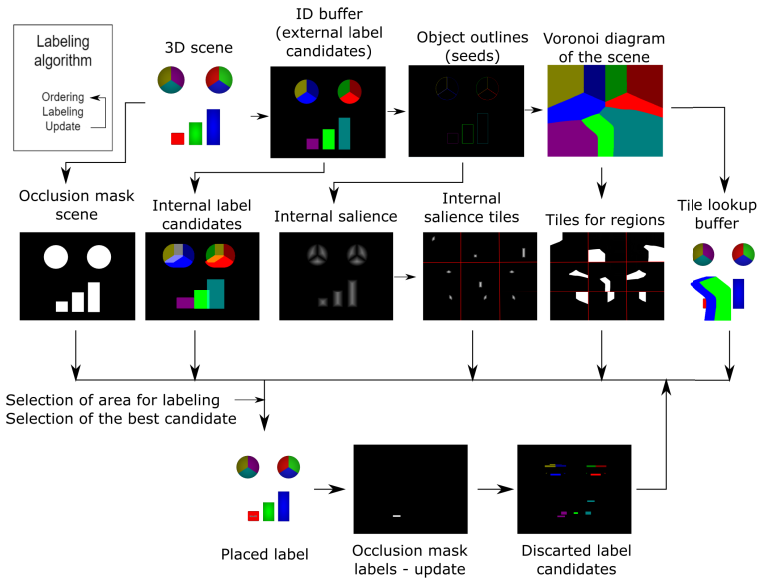


Figure 4.1: Algorithm walk through with buffers involved in separate steps

Figure 4.1 depicts the auxiliary buffers for the labeling procedure. Top two rows depict the preprocessing phase when all the necessary information

Result: Positions of the labels for the given scene
$\forall A_i,\ i \in \{1, ..., n\}$ establish both the internal and the external label candidates;
$\forall x_i \in \bigcup_{AI_j}$ evaluate fitness $F(x_i)$;
$\forall x_e \in \bigcup_{A_j}$ evaluate fitness $F(x_e)$;
while *There is an unlabeled area left in* $\mathbb{A}$ **do**
  Apply occlusion masks, i.e., C_5 and C_6;
  $\forall A_i,\ i \in \{1, ..., n\}$ calculate $C(A_i)$;
  Select $A_{next} = \arg\min_{A_i \subset \mathbb{A},\ A_i \text{ not labeled}} C(A_i)$;
  Calculate $l_{next}^{optI} = \arg\max_{l_i \in AI_{next}} F(l_i)$;
  if $F(l_{next}^{optI}) < t.$ **then**
    Calculate $l_{next}^{optE} = \arg\max_{l_i \in A_{next}} F(l_i)$;
    if $F(l_{next}^{optE}) == 0$ *or* $A_{next} == \emptyset$ **then**
      Discard l_{next}^{optI};
    else
      Keep l_{next}^{optI} based on user preference;
    end
  end
  if *label was placed internally* **then**
    Discard the external label candidates of $A_j, i \neq j$ where the candidate leader line would interset the placed label.;
  else
    Discard the internal label candidates of $A_j, i \neq j$ where the candidate would interset the leader line of the placed label.;
  end
end

Algorithm 2: Algorithm for the mixed labeling in pseudo code

is precomputed. In this whole section, I consider a simple 3D scene for demonstration purposes.

First, I compute the occlusion mask of the scene, encode it as a summed area table, and store it in the *occlusion mask buffer*. Together with the scene, I required the *id buffer* as an input. It represents the external label candidates. I calculate internal label candidates from it and store it in the *internal candidates buffer*. From the *id buffer*, I also calculate the outlines of the objects. Subsequently, I use them to calculate the Voronoi diagram of the scene and store it in the *Voronoi buffer*. Simultaneously, with the calculation of the Voronoi diagram, I calculate the internal salience for all areas A_i. I distribute the salience into tiles based on the id from the *id buffer*, encode it as a summed area table and store it in the *internal salience buffer*. Note that they are linked by the position in the buffers. I do the same for the Voronoi diagram and store it in the *voronoi tile buffer*. I also compute a structure to reduce the number of lookups in the tiles and store it in the *lookup buffer*.

After this preprocessing phase is complete, I evaluate the criteria and store them in *equilibrium buffer*. Note that overlap criteria are evaluated later

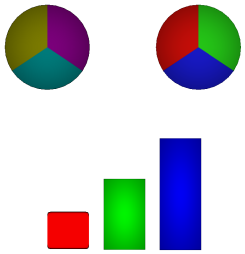
in the calculation. After this, I select an area A_i with the lowest capacity and store its id in the *minsum buffer*. Subsequently, I choose the best label candidate from A_i, respectively AI_i, and store it in the *placed buffer*. After this, I update the occlusion mask for labels, which is stored in *occlusion mask labels buffer*.

Furthermore, I eliminate the internal label candidates, which would intersect the leader line of the placed label and the external label candidates with leader lines that would intersect any placed internal label. The candidates are not discarded but rather masked out. I store this mask in the *intersection buffer*.

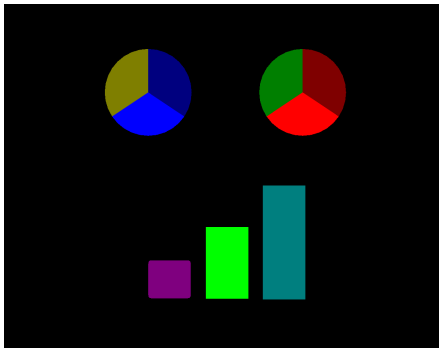
The mixed labeling algorithm is implemented in the class *MixedLabeling.java* in package *tiger.effects.labeling*.

4.3 Determining the label candidates

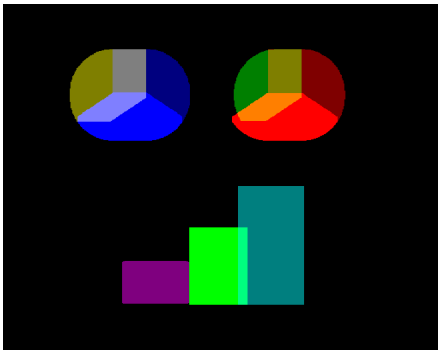
The first step of the algorithm is to determine the label candidates. I consider a simple scene, as depicted in Figure 4.2a. I also receive *id buffer*, depicted in Figure 4.2b, as an input.



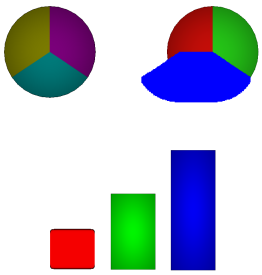
(a) : Scene



(b) : External candidates buffer (Id buffer)



(c) : Internal candidates buffer



(d) : Internal candidates for one object in context of the scene

Figure 4.2: Internal and external label candidates visualized.

External label candidates are thus established since they are represented by the *id buffer*. Internal label candidates are also calculated from the *id buffer*. I do this by the method of jump flooding, as I discussed in section 3.4.2. To obtain the internal label candidates (AI_i) from the external ones (areas A_i), I must spread them by the width of the label left and by the height of the label down. It can happen that the intersection of two areas is not empty $A_i \cap A_j \neq \emptyset$. Therefore candidates on identical position might be propagated a different distance based on their id.

Result: Internal label candidates
Input: *id buffer*, List *ld* of dimensions of label d_i
$wCumul \leftarrow 1$;
$hCumul \leftarrow 1$;
while *ld not empty* **do**
  Create idMask where all the labels are present;
  $d_{minW} \leftarrow$ label dimension with the lowest width from *ld*;
  Do one complete jump flooding pass, propagate idMask to distance $d_{minW} - Wcumul$;
  $Wcumul \leftarrow d_{minW}$;
  remove ids of labels with width lower or equal to $Wcumul$ from idMask and from the list of label dimensions;
end
Do the same for height;

Algorithm 3: Internal label candidates calculation

Algorithm 3 functions in two phases. The horizontal phase propagates the label candidates left by the with of the corresponding label. After it is finished, the vertical phase propagates candidates down by the height of the corresponding label. Figure 4.2c depicts the internal label candidates. It is apparent, that the candidates for the area with the green id were propagated less than the other ones. This is the case because the text of the corresponding label is shorter. Figure 4.2d shows the internal label candidates for one area in the context of the scene.

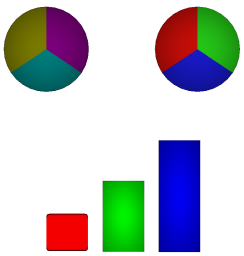
Functionality for determining the internal label candidates is implemented in the class *Erosion5.java* of package *tiger.effects.distance*.

4.4 Evaluating the criteria

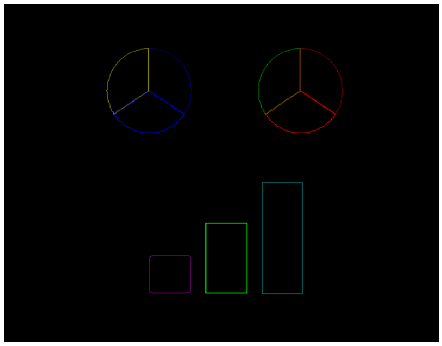
After establishing the label candidates, I must evaluate their fitness with fitness function F. While all the criteria that a given candidate is subjected to are evaluated, not all of them are evaluated at the same time. Some are precomputed and some are evaluated when necessary. Every criterion C_i has its own weight w_i to modify its importance, thus affecting the fitness of candidates. Generally, a different candidate is selected with a different configuration of weights. I provide sliders for the user to modify the weight settings and thus create different label layouts.

4.4.1 Label box salience - external

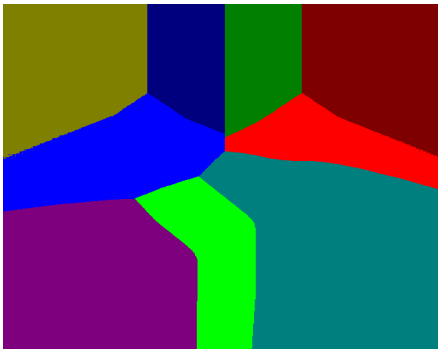
Label box salience in the external areas $\mathbb{S} \setminus \mathbb{A}$ is pivotal, as it measures the ambiguity of placement of the label candidate. It is modeled by the criterion C_1. As I discussed in section 3.5.1, the salience of the label box is dependent on its partitioning among the Voronoi regions of the objects in the scene. Therefore, I must compute the Voronoi diagram of the scene.



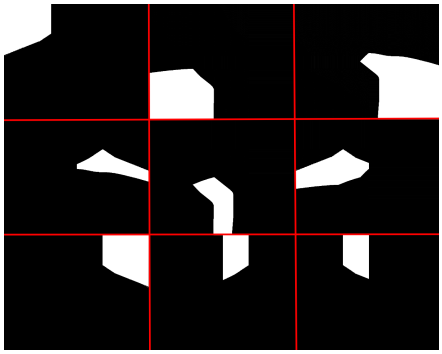
(a) : Scene



(b) : Outlines of the objects in the scene



(c) : Voronoi diagram of the scene



(d) : Tile buffer - each tile is one region of the Voronoi diagram

Figure 4.3: Label salience - external.

Figure 4.3a shows the scene. Figure 4.3b depicts the outlines extracted from the *id buffer*. The outline extraction is done in the fragment shader *Outline.frag* in package *tiger.effect.labeling*. The function of the shader is simple. It searches for discontinuities in the *id buffer*. If any value of the neighboring pixels differs from the pixel value, the pixel is part of the outline. Outline of what objects it is a part of depends, on which bits in the pixel values differ.

The outline serves as seeds for calculating the Voronoi diagram. I calculate it by jump flooding [28]. The functionality to calculate the Voronoi diagram is implemented in the class *Erosion3.java* in the package *tiger.effects.distance*. The Voronoi diagram of the scene is depicted in Figure 4.3c, and I store it in the *voronoi buffer*.

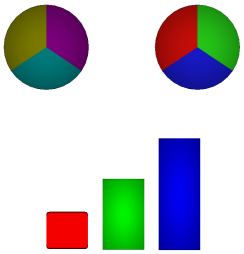
Voronoi diagram solves the problem of the partitioning of the screen space $\mathbb{S}$

into individual regions R_j corresponding to their respective objects. However, partitioning of the label box L_i of any candidate l_i must still be determined. Looking at every pixel of the label box L_i is a possibility, albeit not a fast one to evaluate. I solve this problem by creating tiles, as depicted in Figure 4.3d. Every region R_j corresponds to one tile. Values of the pixels are given by Equation 3.14. I store the tiles in *voronoi tile buffer*. The distribution into tiles is implemented in the fragment shader *tilesDataCopy.frag* in the package *tiger.effects.labeling*. I encode the tile buffer as a summed area table. Now it takes only four lookups in each tile to determine the area of the overlap of L_i with R_j.

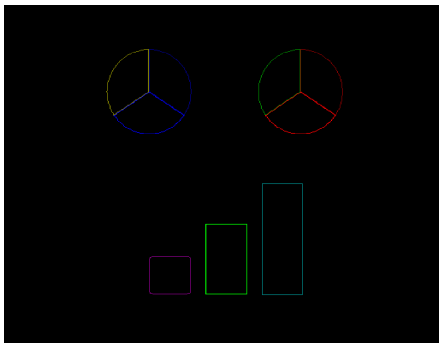
However, an unnecessarily large number of tiles has to be accessed in order to obtain the partitioning of the label box L_i. In order to reduce the number of tiles that need to be accessed, I compute a lookup buffer and store it in the *lookup buffer*. I do this by taking the width w_{max} of the widest label and height h_{max} of the highest label and propagating ids in the Voronoi diagram down and left. This is implemented in the class *Erosion4.java* in the package *tiger.effects.distance*. The algorithmic idea behind calculating the lookup buffer is identical to that of Algorithm 3. It could even be applied to solve this case by introducing a single fictitious label with dimension $d = (w_{max}, h_{max})$ and placing it in the list of label ld, while replacing the *id buffer* with the *Voronoi buffer*. The value of a pixel in the *lookup buffer* represents the ids of the regions R_j a label candidate l_i could possibly overlap if positioned with its lower left corner over the pixel. By reading this value first, I can prevent a significant number of subsequent reads from the *voronoi tile buffer*.

4.4.2 Label box salience - internal

The salience of internal areas, by this I mean area $a \subset \mathbb{A}$, is the method I utilize to assess the ambiguity of the internal label box candidates. It is modeled by the criterion C_2, as I described in section 3.5.1. This is possible by the transition from the salience of a point to the salience of an area.

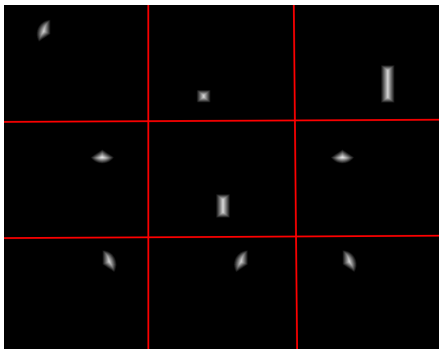


(a) : Scene



(b) : Outlines of the objects in the scene

(c) : Internal salience buffer



(d) : Salience tile buffer - every tile contains salience of anchors for some object

Figure 4.4: Label salience - internal

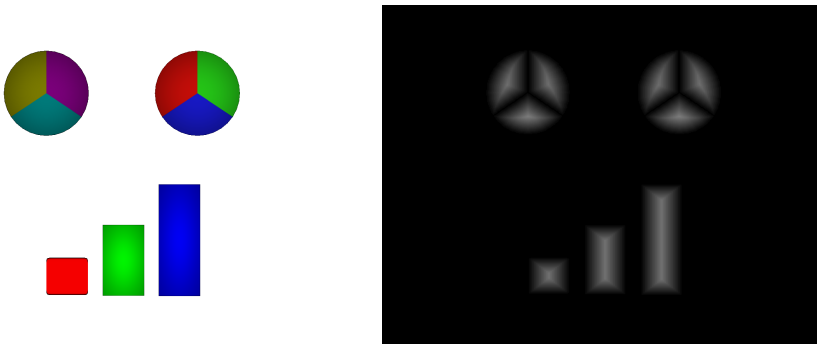
Figure 4.4a depicts the example scene, while Figure 4.4b depicts the outlines obtained from the *id buffer* identically to the previous section. I calculate the point salience while calculating the Voronoi diagram, since during the calculation, I need to store for each pixel not only to which region R_j it belongs to, but also the distance to the closest seed of the outline. For every pixel, this distance represents its salience. Figure 4.4c depicts the salience of individual points. I store the distance data in the *distance buffer 1*.

However, I am interested only in the salience of internal pixels,i.e. pixels $x \in \mathbb{A}$, so I set the pixel values $x_e = 0 \; \forall x_e \in (\mathbb{S} \setminus \mathbb{A})$. I face a similar problem to that of the previous section. I must partition the label box L_i into areas of individual objects A_j. Again, I can sum up over individual pixels, but this approach is very slow. I distribute the salience into individual tiles based on id from the *id buffer*. Each tile corresponds to some area A_j and contains the

saliences of the points that make up A_j. I encode the tiles as a summed area table and store it in the *internal salience buffer*. Now, for every label box L_i I can determine the salience sum of its pixels in a given area A_j with only four lookups. I can further reduce the number of tile lookups by utilizing the *lookup buffer* from the previous section.

4.4.3 anchor salience

I model the anchor salience by the criterion C_3. I require anchors to be salient points of their respective areas, as I discussed in section 3.5.2. This aids attribution of the label to the object it labels. I already calculated the salience of all points $x \in \mathbb{A}$ in the previous section. Thus the criterion C_3 can be evaluated. Figure 4.5b illustrates the salience of the anchors.



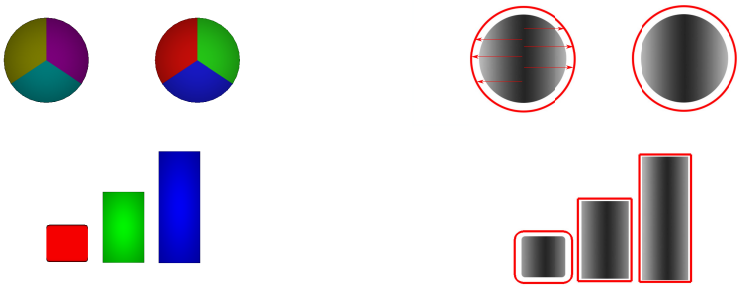
(a) : Scene **(b) :** Anchor saliency

Figure 4.5: Internal and external label candidates visualized.

4.5 leader line length

Leader lines should be as short as possible. This criterion is utilized in many external labeling algorithms [4][11][10]. It aims to make sure that the label and the object it labels are close together. Leader line length can be computed by distance transform (calculating the Voronoi diagram with a different metric). This is done by the method of jump flooding, to produce the result as fast as possible. The implementation is located in the class *Erosion2.java* in the package *tiger.effects.distance*.

However, the external labels are located on the extended silhouette. Length of the leader line is measured from the anchor point to the point on the extruded silhouette of the model. Therefore, I cannot use the outline extracted from the *id buffer*, but it must be extruded first. I store the result in the *distance buffer 2* and utilize it to evaluate criterion C_4.



(a) : Scene

(b) : Leader line length (for horizontal leader lines)

Figure 4.6: Leader line length

Figure 4.6 depicts the scene and corresponding distances to the extruded outline (illustrated in Figure 4.6b). The distance is dependent on the user-selected layout, as the layout affects the metric utilized to calculate the distance transform.

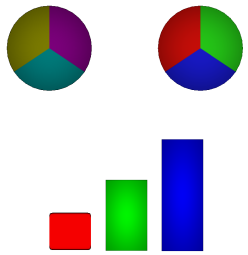
4.6 Overlaps

Labels should not overlap each other, as readability is one of the basic requirements on the label layout [4]. External labels should not overlap even the labeled object. To prevent this, I utilize a pair of occlusion masks. Pixels in these masks carry information about objects and labels present in the scene.

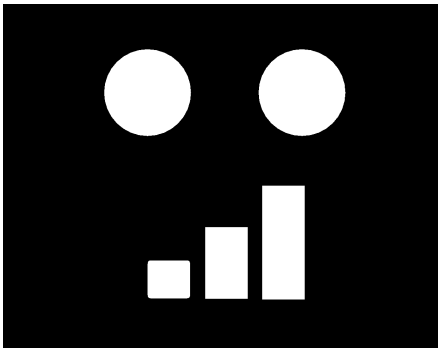
Figure 4.7b depicts an occlusion mask for the scene 4.7a. I calculate it from the *id buffer*. The calculation is straightforward, as a pixel in the occlusion buffer can have only two value - 1 or 0 and the value is 1 for every $x \in \mathbb{A}$,i.e., such pixels in the *id buffer* where the value in *id buffer* is not 0. This is done in the fragment shader *SATDataPrepare.frag* located in the package tiger.effects.labeling.

Furthermore, I encode the occlusion mask as a summed area table and save it in the *occlusion mask buffer*. Overlap of the external labels with any object in the scene can now be determined by 4 lookups in the *occlusion mask buffer*. This is because the label has a rectangular shape. The overlap is detected if the sum of all pixels over the label box is greater than zero. Therefore, the criterion C_5 is efficiently evaluated.

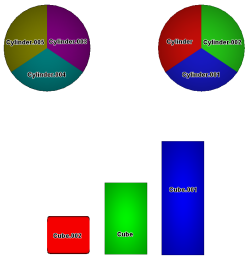
After a label, or more as in Figure 4.7c, is placed in the scene, there is a risk that the subsequent label will overlap it. To prevent this, I introduced the criterion C_6 in section 3.5.5. I write the information about the label position in the *occlusion mask label buffer* 4.7d. The information in the buffer is encoded as a summed area table to allow an efficient label to label overlap evaluation. The *occlusion mask label buffer* starts as a clear buffer and after a label is placed into the scene, it is updated. The update modifies the buffer,



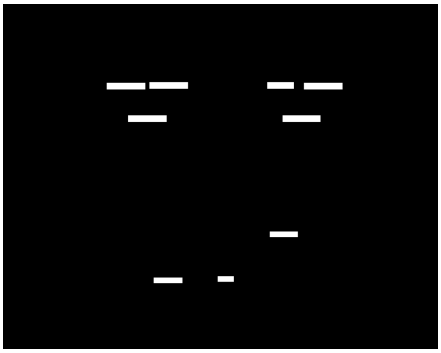
(a) : Scene



(b) : The *occlusion mask buffer* - occlusion mask of the scene



(c) : Scene with the internal labels placed



(d) : The *occlusion mask label buffer*

Figure 4.7: Label to label overlap and label to model overlap masks.

so that the property of the summed area table remains intact, while the information is written to it. This is done in a single pass of the fragment shader *SATupdate.frag* located in the package *tiger.effects.labeling*. There is no need to recalculate the summed area table from the *occlusion mask label buffer*.

The *occlusion mask label buffer* is updated every time a label is placed in the scene.

4.7 Selection of area for labeling

Order in which the algorithm places the labels in the scene is essential. As the hungry optimization I use cannot recover from a bad partial solution, as I mentioned in section 3.6.

I compute the capacity $C(A_i)$ for every area A_i. I select the unlabeled area with the smallest capacity to be the next one, and its label will be positioned in the scene. The idea behind this approach is that areas with small capacities are more susceptible to running out of space where to place the labels, and thus the label candidates. Whereas the high capacity areas will have enough

candidates remaining even if some are no more.

The area capacities are calculated by scattering. Similarly to Scheuermann et al. [31], I perform the bin selection in the vertex shader and configure the hardware blend units to add the incoming fragment values. A bucket corresponds to an area A_i in this case. The fitness function F is evaluated for every label candidate (either in the *id buffer*) or in *internal candidates buffer*. Each label candidate l_i is redirected (by vertex shader) to an address in the *sum buffer* based on the id of area it is a candidate for. The fragment has the value of the fitness function $F(l_i)$ written in its color. Fragments for those label candidates that would cause label to label or,in case of external candidates, label to model overlap are discarded. Blending settings on the GPU ensure that the values of the fitness are added together. The *sum buffer* servers as a set of bins. Each candidate l_i contributes $F(l_i)$ to the bin of its area A_i.

This functionality is implemented in the vertex shader *SphereAreaSum.vert* and the fragment shader *sphereAreaSum.frag* in the package *tiger.effects.labeling*.

After capacities $C(A_j)$ are computed for every area A_j, the unlabeled area A_{next} with the lowest capacity $C(A_{next})$ is selected. This is done by redirecting the capacities of unlabeled areas C_i to one position in *minsum buffer*. In fact, *minsum buffer* has only one address, as no more is needed. Each fragment has the id of the area it represents written in its color component. The depth of each fragment is $\frac{1}{1+C(A_j)}$. And the depth test is responsible for selecting the id of the area with the lowest sum (highest depth).

This functionality is implemented in the vertex shader *minSum.vert* and the fragment shader *minSum.frag* in the package *tiger.effects.labeling*.

4.8 Finding the best candidate

After an area A_{next} with the lowest capacity C_{next} is selected, I must choose its best candidate. I do this by scattering, in a similar fashion to the selection of the area with the lowest capacity. I redirect the candidates into the *placed buffer* at a position corresponding to the id of currently labeled area. The fragment contains the candidate information determining the exact position of the label. Depth of the fragment is corresponding to the fitness F of the candidate. Note, that $F : \mathbb{S} \rightarrow [0, 1]$. Fragments of candidates that would cause overlaps label to label or label to model for external candidates are discarded. Depth test keeps only the best label candidate for area A_{next}. First, I do this for internal label candidates and if the fitness $F(l_{next}^{optI})$ of the best label candidate l_{next}^{optI} is lower than the user set threshold t, then I find the best external label candidate l_{next}^{optE}.

The functionality is implemented by the vertex shader *max.vert* and the fragment shader *max.frag* for external labels and by the vertex shader *maxInternal.vert* and the fragment shader *maxInternal.frag* for internal labels. All shaders are located in the package *tiger.effects.labeling*.

4.9 Overlap elimination

Both Internal and external label candidates which overlap already placed labels and external label candidates that overlap the model were already masked out during the calculation of the area capacities C_j and selection of the best label candidate.

However, there is another undesirable overlap scenario that I must counter. After an internal label is placed in the scene it can eliminate some other external label candidates because it overlaps their future leader line. Similarly, after an external label is placed in the scene, it can eliminate some internal label candidates because they would overlap its leader line.

I remove the candidates for which either of the two cases occurs. This is accomplished by the fragment shaders *leaderlinesIntersection.frag* and *leaderlineIntersection_1.frag* in the package *tiger.effects.labeling*.

4.10 Chapter summary

Chapter 4 describes the implementation of the mixed labeling algorithm. At first, I discuss the technologies utilized to carry out the implementation in section 4.1. In the following section 4.2 I recapitulate an overview of the proposed mixed labeling method. I go on and explain individual steps.

First, I must establish the internal and external label candidates, as discussed in section 4.3. The external label candidates are already stored in *id buffer*, as they are an input of the algorithm. For every area A_i, I calculate the internal label candidates l_i from the external ones by propagating them left and down by the dimension of the label d_i. I use jump flooding to make the calculation faster. I store the internal label candidates in the *internal candidates buffer*.

Subsequent section 4.4 brings an overview of the methods utilized to calculate the values of the membership functions $C_1, ..., C_6$ for any given candidate.

After the label candidates are established, I evaluate their fitness and select an unlabeled area A_{next} with the lowest capacity $C(A_{next})$ in section 4.7. I do this by scattering [31].

In the following section 4.8. I describe the process, how the best candidate is selected. I select the best internal label candidate l_{next}^{optI} for A_{next} from AI_{next}. If the fitness of the candidate $F(l_{next}^{optI})$ is lower than the user defined threshold t I find the best external label candidate l_{next}^{optE}. I place the best candidate in the scene.

The final section 4.9 discusses problems connected to the label to leader line overlap. After the best label candidate is placed in the scene, some of the remaining label candidates are no longer viable. The algorithm iteratively places the labels in the scene until there are no unlabeled areas A_i left. I give the user the option to set weights of the criteria and affect the label layout through the process.

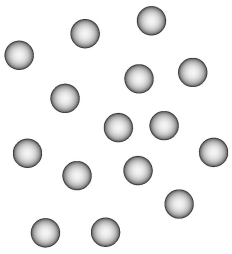
Chapter 5

Results

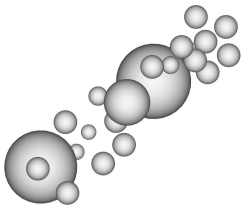
I present the results obtained with the algorithm in the following section. First, I present models which I will subject to testing in section 5.1. Some of which are on a simple side. Nevertheless, they Illustrate the possibilities of the mixed labeling algorithm well, thus are beneficial to include.

5.1 Tested models

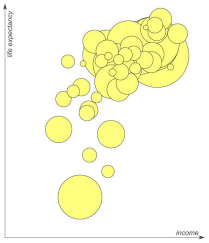
I included six models in total. They are depicted in Figure 5.1. The first two, Figures 5.1a and 5.1b, are model scenes to test the behavior of the algorithm and effects of the criteria and their weights.



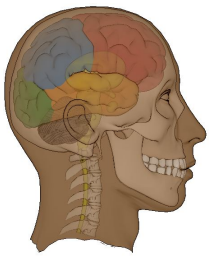
(a) : Spheres



(b) : Complex spheres



(c) : Gapminder



(d) : Head

(e) : Wheel fork

(f) : Digestive system

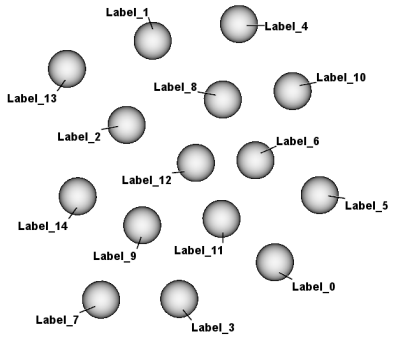
Figure 5.1: Models used for testing.

Figure 5.1c depicts a visualization based on data from gapmider.org, which examines the dependency between income and life expectancy. Figures 5.1d,

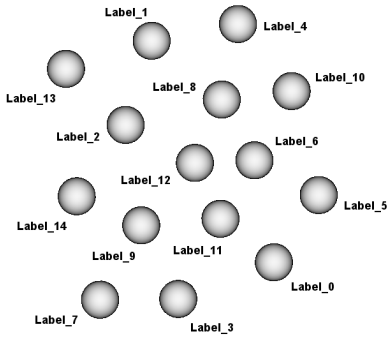
5.1e and 5.1f depict 3D models. These are included to demonstrate that the proposed algorithm can deal with the labeling of semi-transparent objects.

5.2 Labeling results

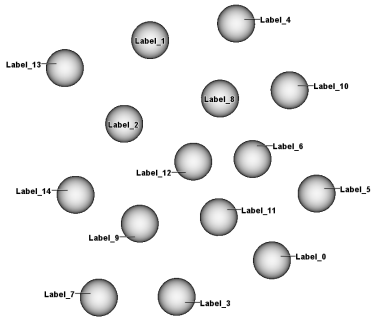
The first model comprises of fifteen spheres. I include this scene mainly to demonstrate capabilities of the algorithm in terms of labeling objects externally without the use of the leader lines.



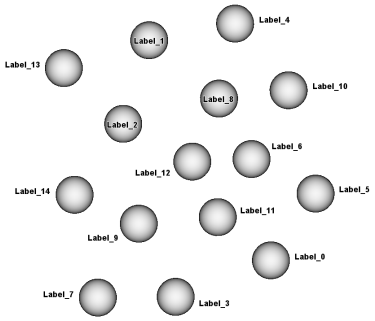
(a) : External with leader lines



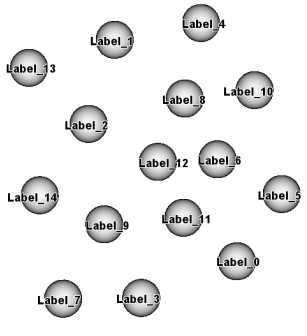
(b) : External without leader lines



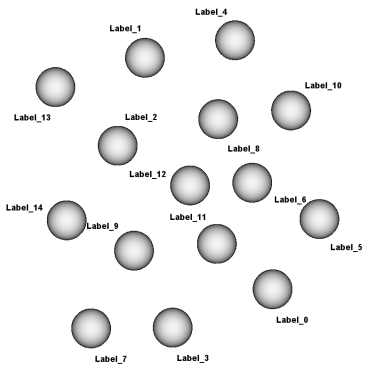
(c) : Mixed with leader lines



(d) : Mixed without leader lines



(e) : Internal



(f) : Ambiguous labeling

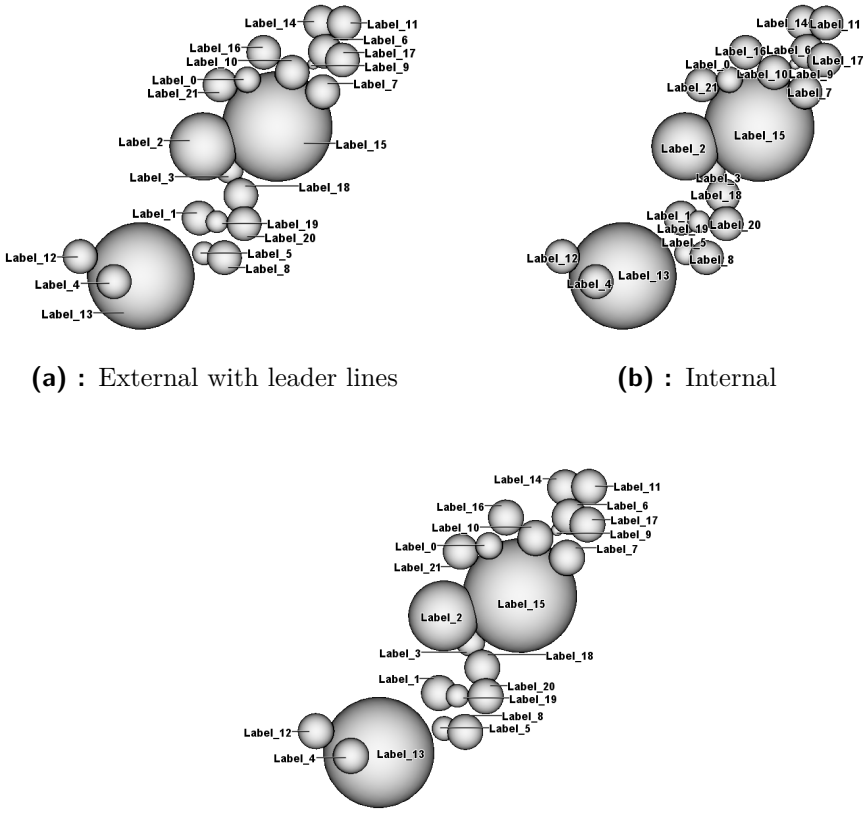
Figure 5.2: Results for the testing scene with a set of spheres.

Figure 5.2a shows strictly external labeling. Label boxes maximize the criterion C_1 and position themselves in positions in the scene, where they are easily attributable to their respective objects. Maximizing criterion C_1 is in this case equivalent to minimizing ambiguity, as it should be. Figure 5.2b depicts the same scene, but without leader lines.

Figures 5.2c and 5.2d depict mixed labeling of the scene with and without the use of the leader lines, respectively. As the spheres are almost identical, the minuscule differences in fitness function F for individual labels, which cause them to be placed internally or externally, can be attributed mainly to the limited screen resolution. If I move the model around the screen it causes different labels to be placed internally.

However, this is due to the fitness values for the individual candidates being very close to the threshold t. The mixed labeling algorithm attempts to place the internal labels first. The external labels are placed only on the basis of internal candidates fitness being lower than the threshold t. Unfortunately, this poses a certain limitation on the algorithm. Some of the external labels clearly occupy, even if ever so slightly, more ambiguous locations than others. It would benefit the layout if these labels would be placed internally instead. Namely, the sphere in the center of the scene could have their labels internal. Figure 5.2e depicts all of the labels placed internally. This is a suboptimal solution, in my opinion, as the labels occlude the spheres. However, there are not many features of the spheres worth examining.

Figure 5.2f depicts an example of the scene, where I produced ambiguous labeling by inverting the value of criterion C_1, i.e., taking $1 - C_1$. Labels 6, 8, 9, 11 and 12 are positioned in ambiguous areas. We can attribute them to the correct object with some effort. However, this should not be the case for any illustration which uses labels.



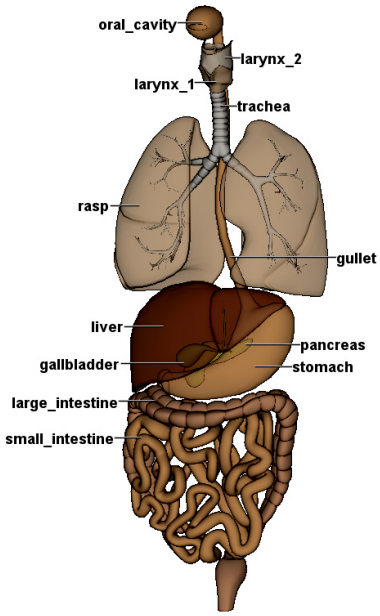
(a) : External with leader lines

(b) : Internal

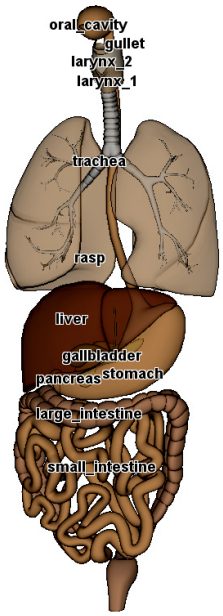
(c) : Mixed with leader lines

Figure 5.3: Results for a testing scene with a set of spheres of different sizes.

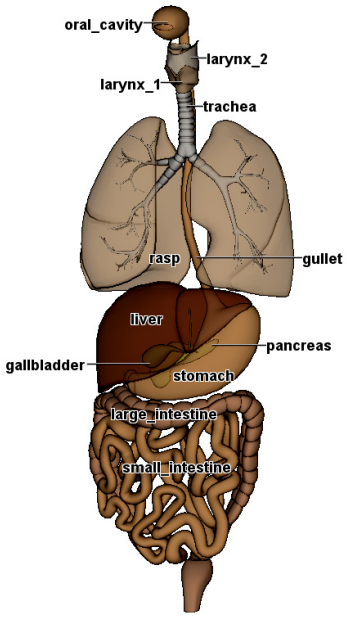
Figure 5.3 shows a scene with spheres of various sizes. This scene is more complicated than the previous one. Figure 5.3a manages to label all of the spheres and it is possible to attribute all of the labels to the correct objects. In case of all labels being internal, depicted in Figure 5.3b, it is challenging to associate the labels with the objects and it is hard to be certain about the matching. While we can easily identify the labels for the large spheres, the same is not the case for the smaller ones, where labels are positioned rather densely. Mixed labeling in Figure 5.3c combines the best of both. We can associate every label to an object, be reasonably certain about it and utilize some additional positions not available for strictly external labeling.



(a) : External with leader lines



(b) : Internal

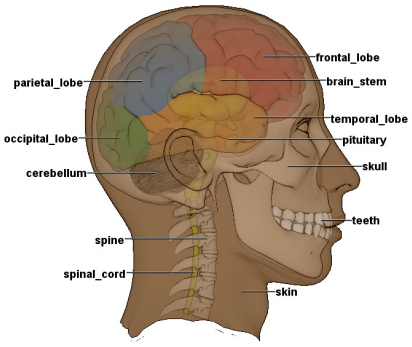


(c) : Mixed with leader lines

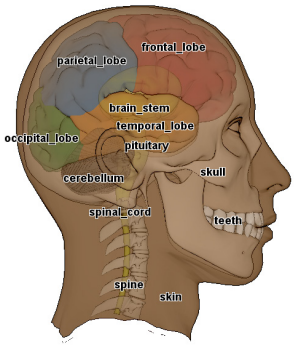
Figure 5.4: Results for scene containing the model of the digestive system.

Figure 5.4 depicts a model of the digestive system. The external label layout, depicted in Figure 5.4a, is very well organized and we can attribute every label without difficulties. In contrast, the internal labeling in Figure 5.4b does not exhibit identical properties. It is not clear, to which objects some of the labels belong. Some objects are occluded by the labels.

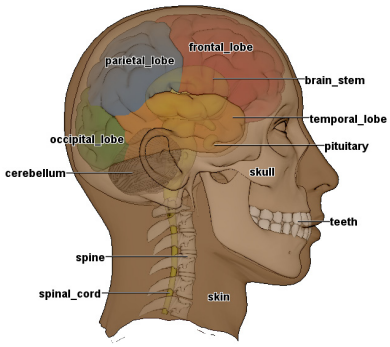
However, labels of large objects can be assigned with ease, As in the case of the previous Figure 5.3. Figure 5.4c combines the two. We can assign every label to the correct object, as the small objects are labeled externally. We also managed to increase the compactness of the illustration, which is one of the requirements on labeling, according to Ali et al. [4].



(a) : External with leader lines



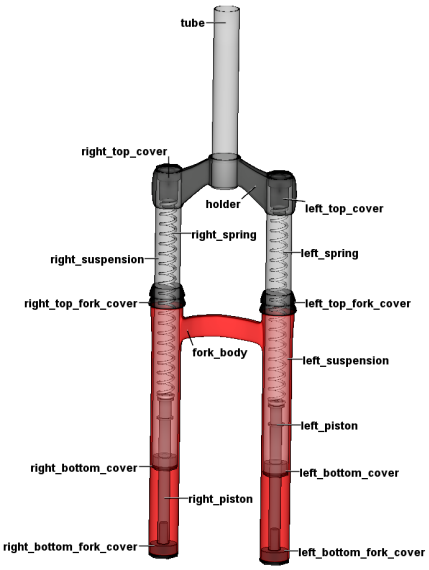
(b) : Internal



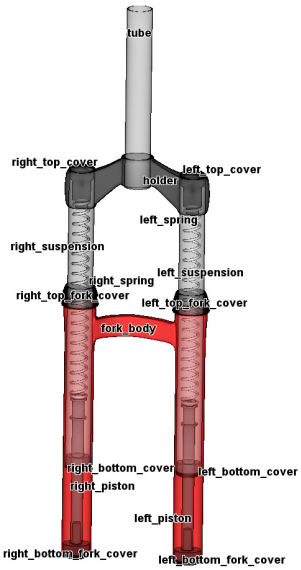
(c) : Mixed with leader lines

Figure 5.5: Results for scene containing the model of the human head.

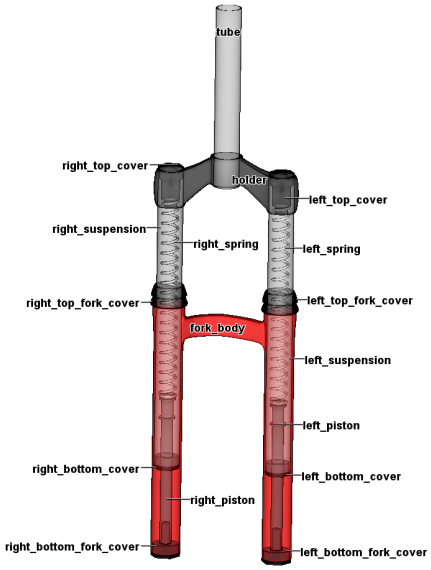
Figure 5.5 depicts the human head. Even though it is a model that is rendered utilizing semi-transparency, the internal labeling in Figure 5.5b is reasonable. Only pituitary and spinal cord are hard to assign. It this case, we can deduce what do the labels mark, because we are already familiar with the objects. External labeling, depicted in Figure 5.5a solves the problem of the small objects. Figure 5.5c shows the mixed labeling of the head. Only those internal labels which are clearly attributable to their objects are retained from the internal labeling. External labels complement them very well, labeling those objects, for which the internal labels would be ambiguous.



(a) : External with leader lines



(b) : Internal

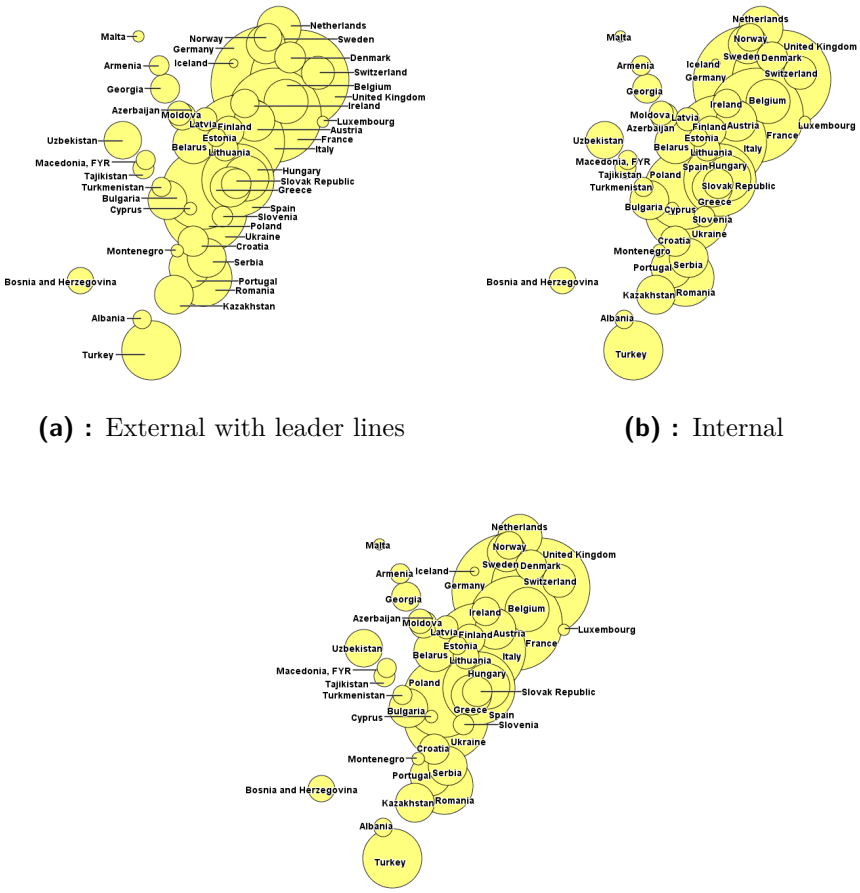


(c) : Mixed with leader lines

Figure 5.6: Results for scene containing the model of the wheel fork.

Wheel fork, depicted in Figure 5.6, is the last of the semi-transparent models. It is the most complex one and is hard to label utilizing any method. We can assign all the labels in the case of Figure 5.6a. However, we must pay very close attention to the positions of the anchors. Internal labels, depicted in Figure 5.6b, fail at the task of labeling the model. There is not enough space for almost any label, apart from the fork_body, holder and the tube. Those are exactly the three labels with internal placement in Figure 5.6c. External labels are almost identical to those in Figure 5.6a. However, the

mixed labeling model utilizes some additional space, which is not accessible to the model with only external labels.



(a) : External with leader lines

(b) : Internal

(c) : Mixed with leader lines

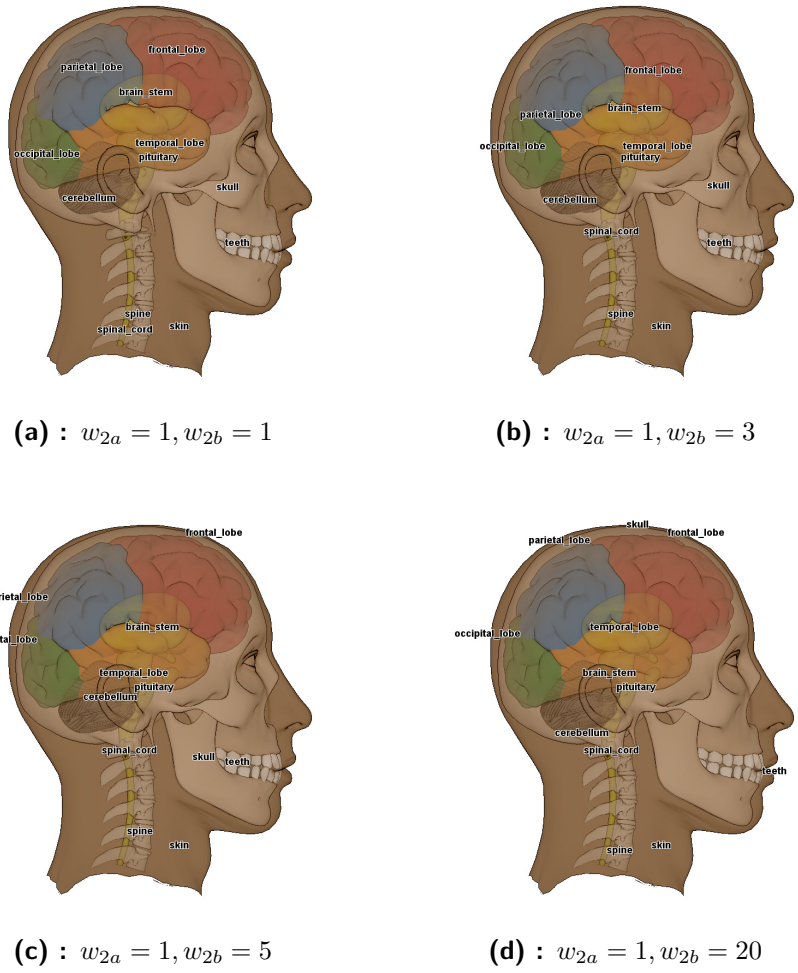
Figure 5.7: Results for scene containing a model of a wheel fork.

Figure 5.7 depicts the labeling produced for an illustration based on the data from the gapminder[1]. If the labels are placed only externally, there is no space left for some of them and the internal label take their place. This scene contains the largest number of labeled objects from the testing scenes.

[1] https://www.gapminder.org/

5.3 Influence of the weights

I show the influence of weights on the label layout the algorithm produces. As there are many possible weight settings, I include only a few examples depicting the function of individual criteria and their weights.

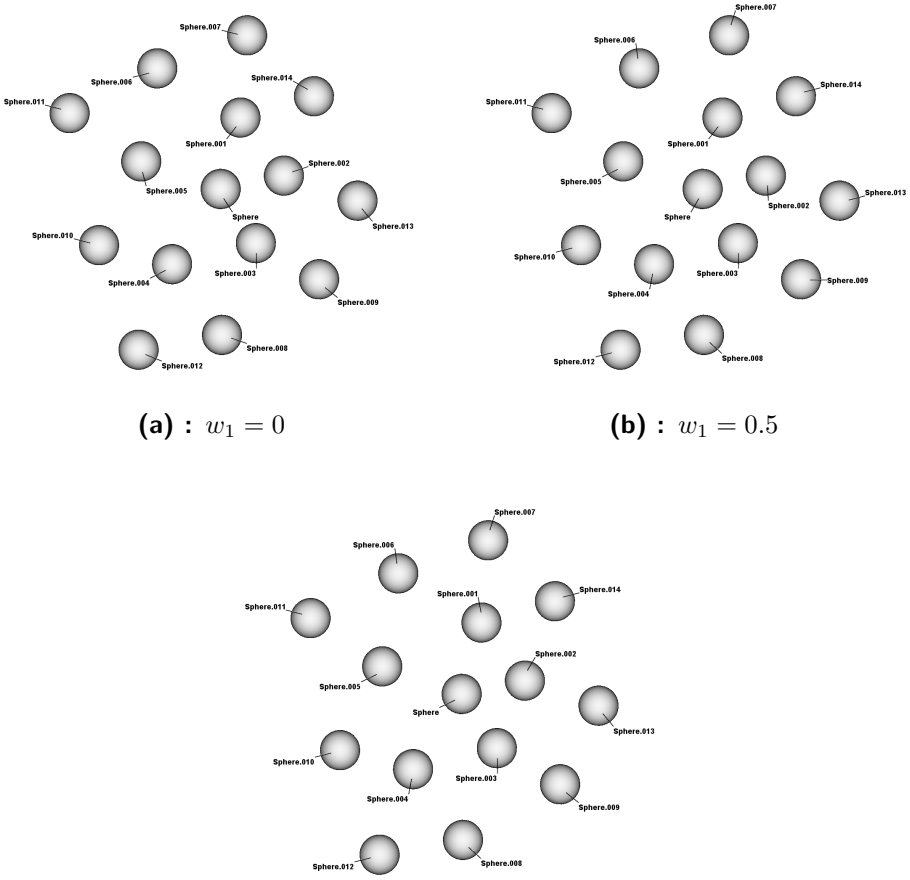


(a) : $w_{2a} = 1, w_{2b} = 1$ **(b) :** $w_{2a} = 1, w_{2b} = 3$

(c) : $w_{2a} = 1, w_{2b} = 5$ **(d) :** $w_{2a} = 1, w_{2b} = 20$

Figure 5.8: Effects of the weights w_{2a} and w_{2b}.

I can model internal salience of the label, as described by the criterion C_2, as being composed of two components. One that forces the labels to overlap the object they label - C_{2a}, while the other one forces them not to overlap the objects they do not label - C_{2b}. Criteria C_{2a} and C_{2b}, with their respective weights w_{2a} and w_{2b}, affect the internal label positions as depicted in Figure 5.8. The increasing weight w_{2b} forces the labels to the empty space around the model.

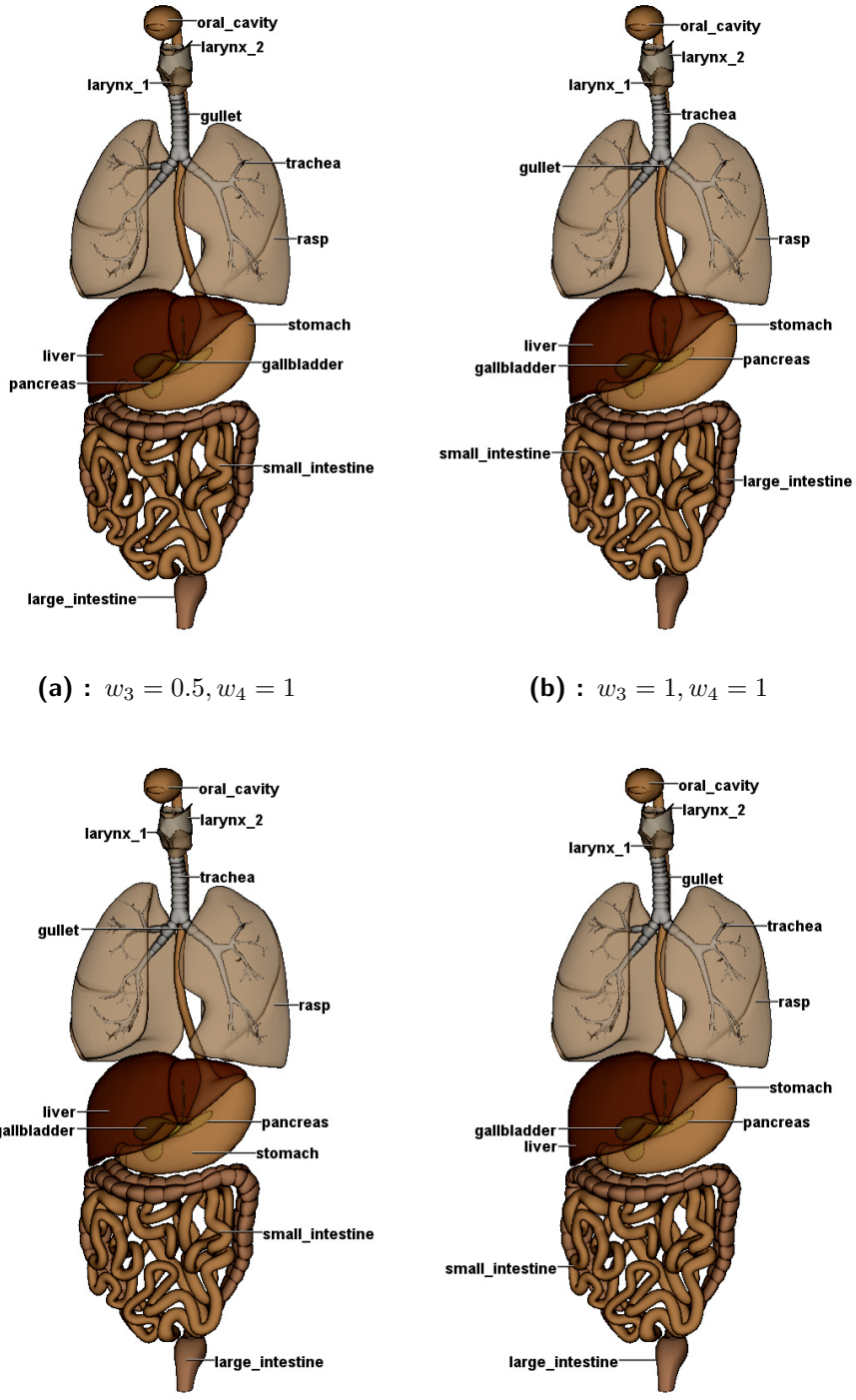


(a) : $w_1 = 0$ **(b) :** $w_1 = 0.5$

(c) : $w_1 = 1$

Figure 5.9: Effects of weight w_2.

Figure 5.9 depicts the effects of the increasing weight $w1$ of the external salience criterion C_1. As the weight increases, the labels are forced into more salient positions.



(a) : $w_3 = 0.5, w_4 = 1$ **(b) :** $w_3 = 1, w_4 = 1$

(c) : $w_3 = 2.5, w_4 = 1$ **(d) :** $w_3 = 0.5, w_4 = 2.5$

Figure 5.10: Effects of weights w_3 and w_4.

Figure 5.10 depicts the effects of w_3 - anchor salience weight and leader line length weight w_4. These two criteria are contradicting each other, as higher salience means longer leader line and vice versa.

5.4 Performance

I discuss the performance of the algorithm in this section. The time the algorithm needs to calculate the layout is an important indication pertaining to both the quality of the design and the quality of the implementation. Furthermore, the render time limits possible use cases for the algorithm. According to Nielsen [25], if the response time is below 100ms, the application gives the results immediately. If the algorithm takes significantly more than that, it is not feasible to use it in interactive applications.

5.4.1 Testing hardware

The performance of the algorithm was evaluated with the following hardware & software configuration.

- **CPU**: Intel Xeon W-2125 @ 4.00GHz, 4cores, 8.25MB L3 cache
- **GPU**: NVIDIA TITAN Xp, 12GB of GDDR5X RAM, 3840 unified shaders, 240 texture units
- **RAM**: 64GB DDR4 @ 2667MHz
- **OS**: Windows 10 Pro 64-bit
- **Java**: JDK 1.8
- **OpenGl**: OpenGL 3.1
- **OpenGl**: version 3.30 compatible

Resolution of the windows was 800×600 for every scene tested and garbage collection in JVM suppressed by setting *-Xmx16g -Xms16g*.

5.4.2 Performance results

The Algorithm was tested for scenes of various complexity, containing between 6 and 46 objects that were supposed to be labeled. I described the testing hardware in the previous subsection 5.4.1. Figure 5.11 depicts the average time needed to place label a for all of the objects in the tested scene. The lower error bars are for all the labels placed internally. Meanwhile the upper error bars are for all the labels placed externally.

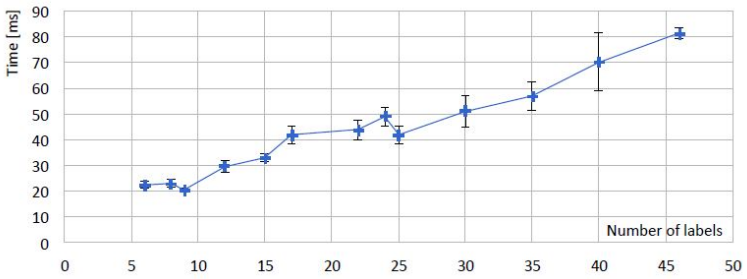


Figure 5.11: Average time required to calculate the label layout based on the number of labels objects.

The algorithm requires less than 100ms for all tested scenes. According to Nielsen [25], the results can be considered as obtained almost immediately. Therefore, the algorithm is suitable even for use in interactive applications. Even significantly weaker hardware can suffice for response times of less than 100ms for models with a small number of parts.

5.5 Chapter summary

I presented the results that the mixed algorithm produces in this chapter. The positions of the labels are vastly affected by the user-set weights of individual criteria, which in turn modify the fitness $F(l_i)$ of individual label candidates. I described the influence of weights on the label layout in section 5.3. Section 5.4 describes the performance of the algorithm. It was able to place labels under 100ms for all labels and thus is suitable for use in the environment, where interaction with the user is key.

Chapter 6

Evaluation with users

Speed of the algorithm is not the only metric to measure, how well does it perform. Substantially more critical is how the end user perceives the results generated by it. After all, it is the user for whom the various labeling related applications of the algorithm are intended. I designed a user study to test, how do users respond to the label layouts generated by the algorithm.

6.1 Study design

The design of the study is important. It affects, what kind of results I hope to obtain from the study. I propose three main methods of labeling a given scene. The labels can be placed in three following fashions:

- **M0: Internal** - all labels are placed internally
- **M1: External** - all label are placed externally
- **M2: Mixed** - some labels are placed internally, while the rest is placed externally

I would like to see whether the users perceive any differences between the three methods. Therefore, I measured the two following variables.

- **Time** t - How long did it take for the participant to select a label for a designated part of the model
- **Error** e - How often do participants make a mistake

Participants were shown a Likert survey after finishing each model to collect more data relevant to their perception. They chose answers to the following statements:

- **easiness** - I found the label layout well arranged.
- **confidence** - I was able to quickly assign a label to the correct part.
- **speed** - I was sure when assigning the labels to the parts.

Participants had standard Likert scale choices:

- Strongly agree
- Agree
- Neither agree or disagree
- Disagree
- Strongly disagree

Likert scale base questionnaires are suitable in this case because they allow for statistical evaluation. Therefore, I can make statistical conclusions about the opinions of the participants on the labeling methods.

I will utilize confidence intervals to evaluate whether there are statistically significant differences among the samples means, i.e., whether the participants make fewer errors with one method than with the others or whether one the method is faster.

If the confidence intervals for different methods do not overlap, then there is a statistically significant difference between the sample means. I also calculate the confidence interval for the difference between the samples. If it does not contain zero, then there is a statistically significant difference between the sample means.

I utilize different confidence intervals to evaluate the statistical significance of the results for every variable. I transform the number of errors for each model and method into error rates by the LaPlace method, recommended by Lewis and Sauro [23]. I calculate the confidence intervals for the error by adjusted Wald intervals. This approach is recommended by Agresti and Coull [3]. In case of confidence intervals for time, I calculate them as confidence intervals for task completion, as suggested by Sauro and Lewis [30]. I will evaluate the results from the Likert surveys utilizing the t-distribution. It provides the best confidence interval regardless of the sample size [22].

I will conduct the statistical evaluation with the confidence level $1-\alpha = 95\%$, where α is the level of significance, in my case $\alpha = 5\%$.

I decided to use the **between-subject design**, as its advantages outweigh the disadvantage. I must test with a large enough number of participants so that the results of the study carry weight.

Since I plan on testing with a lot of users, I opted for remote testing. The data from the participants will be collected by a web service designed specifically for the collection of the data for the purposes of experiments involving users. The application was created by Antonina Lebedeva [22] and is free for anyone to use. I will use already running service instance of it, called Sfix[1]. .

[1]Sfinx is currently [May-20-2019] running at address https://sfinx.felk.cvut.cz/

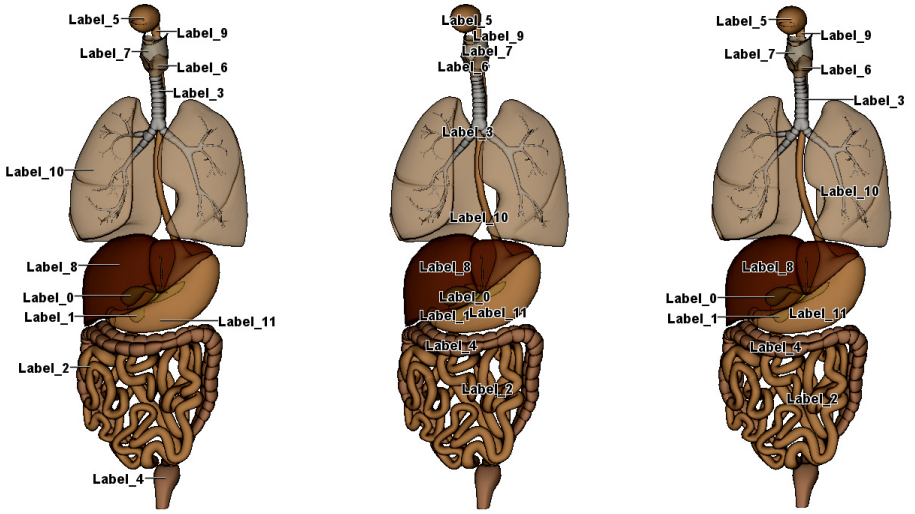
Sfinx is a web service for collecting data from empirical studies. It allows for the collection of various kinds of data and automatically processes them utilizing confidence intervals [8].

The experiment itself consisted of 3 parts. At first, the participant is presented with an introductory web page containing the description of what to expect from the experiment. I require gender and age from the participant. After the participant provides the requested information and submits the form, the first model is displayed. It is only a trial model so that the participant has a chance to get familiar with the controls.

After the participant clicks on the *start* button, one part of the model is highlighted in light green. The participant clicks on the label, which is marking the part in his/her opinion. If the participant thinks there is no label for the part, he/she clicks on the *no label* button. Similarly, in case of a split decision, he/she clicks the *I cannot decide* button. Upon a click on the label or one of the buttons, there is a one-second delay, and the next part is highlighted. After every part of the trial model has been highlighted, the participant is prompted to continue to the user study. The participant is then shown the subsequent model, three models in total. The walkthrough is identical to that of the test model. For each model, the time it took the participant to click on the labels is recorded together with mistakes participant made. After the participant completes any model, A Likert survey follows. The participant fills out the survey and continues to the next model until the test is finished. Models which I used for the testing are included in the following section.

6.2 Testing models

The test was conducted on three different models with one introductory model so that the participants can familiarize themselves with the controls. The model of the digestive system, depicted in Figure 6.1, was shown to the participants first. They could try how the controls of the test environment respond to their inputs and get to know the testing procedure.



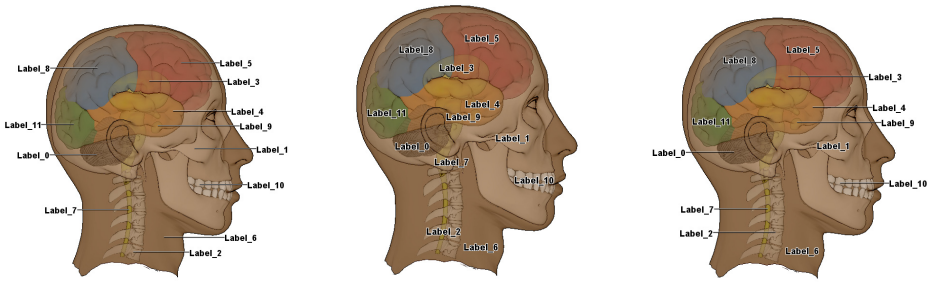
(a) : External with leader lines

(b) : Internal

(c) : Mixed with leader lines

Figure 6.1: The introductory scene for the user testing, containing the model of the digestive system.

The first testing model was the head, as depicted in Figure 6.2.



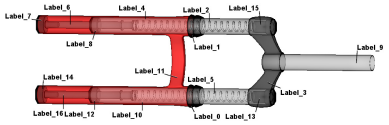
(a) : External with leader lines

(b) : Internal

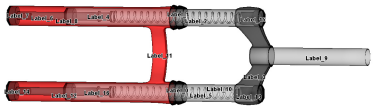
(c) : Mixed with leader lines

Figure 6.2: The first test scene for the user testing, containing the model of the human head.

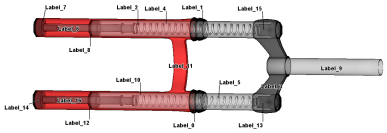
The second testing model was the wheel fork, as depicted in Figure 6.3.



(a) : External with leader lines



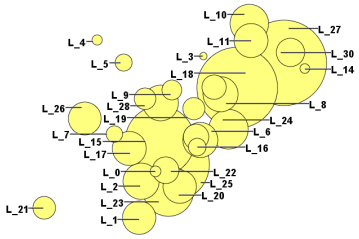
(b) : Internal



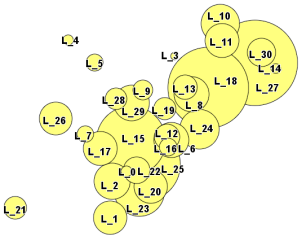
(c) : Mixed with leader lines

Figure 6.3: The second test scene for the user testing, containing the model of the wheel fork.

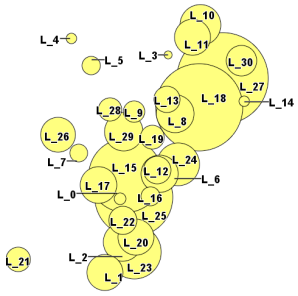
The third, and the last, testing model was the gapminder, as depicted in Figure 6.4.



(a) : External with leader lines



(b) : Internal



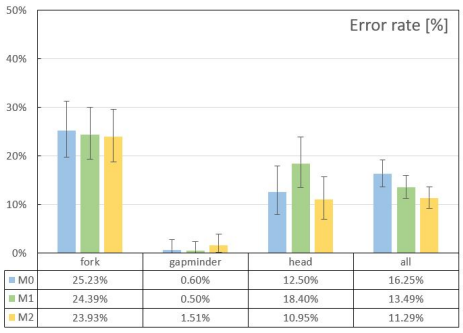
(c) : Mixed with leader lines

Figure 6.4: The third test scene for the user testing, containing the visualization of dependency between income and life expectancy.

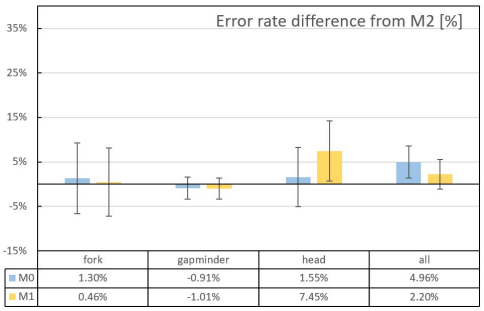
6.3 Results of the study

I discuss the results of the study in this section. When comparing methods, I consider the mixed labeling (method **M2**) to be the baseline approach, as the combination of the internal and external labels is the main focus of this thesis.

I conducted the study, as I described in section 6.1. Total of 63 participants (mean age: 27.9 years, SD: 10.6 years) took part in the experiment — all regular users of computers.



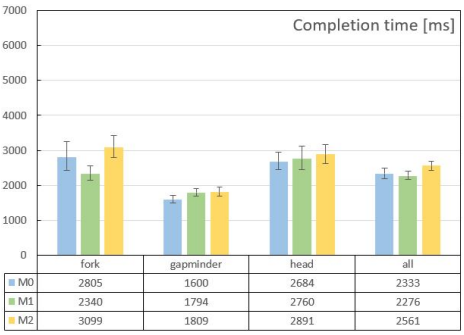
(a) : The number of errors per model and method



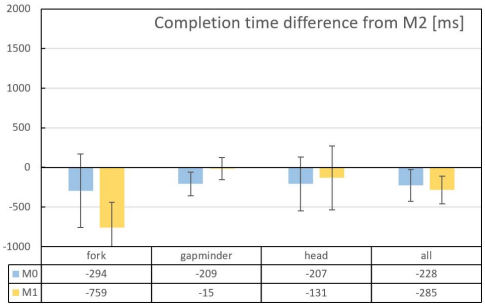
(b) : The number of errors compared with the method **M2**

Figure 6.5: The number of errors

Figure 6.5a depicts percentage of errors made by the users on individual models, as well as overall. The participants made a lot of mistakes when faced with the model of the wheel fork. In contrast, the gapminder illustration posed no problems for the participants. The semi-transparency of the models likely contributed to the outcome. Figure 6.5b depicts differences in error rate compared to methods **M2**. The mixed labeling (method **M2**) performed better overall. The difference was statistically significant when compared with the method **M0**.



(a) : Time to click one label per model and method

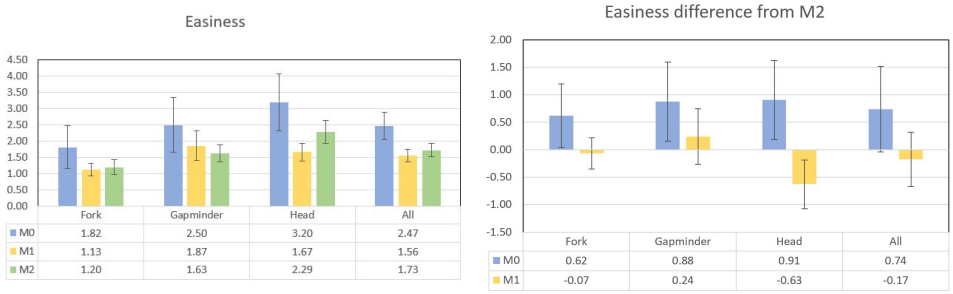


(b) : Times compared with the method **M2**

Figure 6.6: Time to click one label

The time it took for the participants to select one label is depicted in Figure 6.6a for every model. Overall, the method **M2** performed worse than the remaining two. The result is statistically significant, as zero is not part of

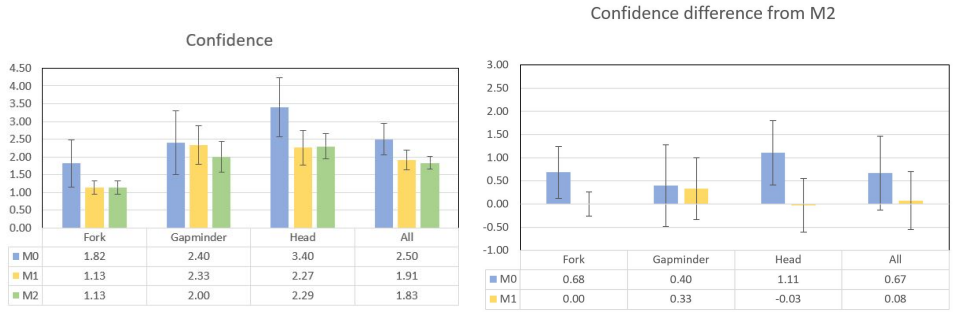
the confidence interval for the difference for all models, as depicted in Figure 6.6b.



(a) : Per method and model **(b) :** Compared with the method **M2**

Figure 6.7: Easiness of attributing a label to the correct part

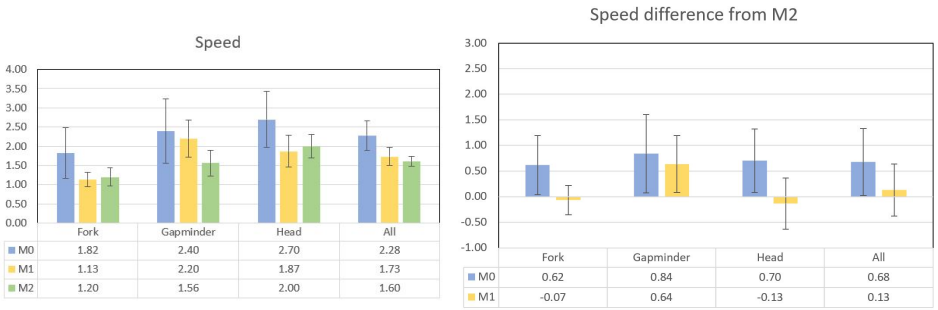
Figure 6.8a depicts the opinions of the participants on how easy it was to assign a label to a given part. Method **M0** (internal labeling) performed worse than method **M2** (mixed labeling). However, the difference is not statistically significant.



(a) : Per model and method **(b) :** Compared with the method **M2**

Figure 6.8: Confidence in attributing the label

Overall, the difference in the confidence of the participant in selecting the correct label for a given part was not statistically significant, as shown by Figure 6.8. However, participants were not confident when attributing the internal label for semi-transparent models. And the confidence of the participants is significantly worse for the internal labels compared with the mixed labels in this case.



(a) : Per model and method **(b) :** Compared with the method **M2**

Figure 6.9: Perceived speed of attributing the label

Figure 6.9 depicts the results of the survey for the question of speed. Overall, no method is statistically worse than the method **M2**. However, participants felt that assigning a label to the corresponding part takes them significantly more time in case of the head and the wheel fork.

6.4 Chapter summary

I designed and conducted a user study with a total of 63 participants (mean age: 27.9 years, SD: 10.6 years). I described the design of the study in section 6.1 and included the tested models in section 6.2. The study did find a statistically significant difference in error rates between **M0** and **M2**. There was a statistically significant difference in times between the mixed labeling method **M2** and the remaining two. Participants took a longer time to identify the label in case of the mixed labeling scenes. The Likert questionnaires suggest that the participants found internally labeled models to be inferior to those labeled with the mixture of internal and external labels, as shown in Figures 6.7, 6.8, and 6.9. However, no difference in means is statistically significant.

Chapter 7

Conclusion

I pursued a GPU evaluated mixed labeling algorithm in this thesis. The algorithm operates in real time for medium complexity models and is capable of producing label layouts with both internal and external labels. It utilizes a novel approach to evaluate salience of the whole surface of the label box with respect to every object in the scene and their Voronoi regions. Evaluation with users revealed a statistically significant difference in error rates between the layouts labeled only with the internal labels and those labeled with the mixed labeling style.

I described various approaches to the topic of labeling over the long history of the subject in chapter 2. In the next chapter, I described the problem, which I was solving, in more depth. I followed up the description with a general overview of the techniques I utilize to calculate the necessary data structures. Afterward, I presented an overview of the algorithm and described the design of the individual steps of the algorithm in the subsequent sections. Chapter 4 mirrored the previous chapter but viewed the algorithm from the perspective of the implementation. I described what information I stored in the distinct buffers and how I utilize it in the algorithm.

I presented various label layouts which I generated with the algorithm in chapter 5. In total, I used six models for the demonstration of the functionality of the algorithm. I demonstrated how the assessment of external label salience improves the label layouts and contributes to unambiguous label positions. Thus the labels do not need to be connected with leader lines in scenes, where it is desired. I also measured the performance of the algorithm on models with up to 46 labeled objects. All of the label layouts were generated in less than 100ms.

The last chapter 6 presented the results of the evaluation with the users. I designed a user study to evaluate how the end consumers - the users perceive the various label layouts. I tested scenes with only internal, only external, and mixed label positions. Participants had to correctly assign the labels to their respective parts in the shortest time possible. I collected error rates and completion times together with the data from the Likert questionnaires, in which the users had to rate the label layout. I collected the data on their perception of easiness, confidence, and speed with which they selected the labels. I statistically evaluated all of the collected data. I found that there is

a statistically significant difference in error rates between the internal label layout and the mixed label layout.

7.1 Future work

The algorithm this thesis presented has a large number of possible usages, as it places the labels in a versatile fashion. However, its results can be further improved. Namely, the recognition of the situations when it is possible to place an external label without the leader line confidently can be improved. The algorithm can be further extended for the use in scientific application such as document visualization. The number of labels that the algorithm can place is limited as of now. The use of texture arrays could remove this limitation.

Bibliography

[1] Jogl - java binding for the opengl api, 2019. [Online; accessed 20-May-2019].

[2] Pankaj K Agarwal, Marc Van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3-4):209–218, 1998.

[3] Alan Agresti and Brent A Coull. Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126, 1998.

[4] Kamran Ali, Knut Hartmann, and Thomas Strothotte. Label layout for interactive 3d illustrations. 2005.

[5] Michael A Bekos, Michael Kaufmann, Antonios Symvonis, and Alexander Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry*, 36(3):215–236, 2007.

[6] Richard E Bellman and Lotfi Asker Zadeh. Decision-making in a fuzzy environment. *Management science*, 17(4):B–141, 1970.

[7] Marc Benkert, Herman Haverkort, Moritz Kroll, and Martin Nöllenburg. Algorithms for multi-criteria one-sided boundary labeling. In *International Symposium on Graph Drawing*, pages 243–254. Springer, 2007.

[8] Ladislav Čmolík. Sfinx, 2017. [Online; accessed 20-May-2019].

[9] Ladislav Čmolík. Tiger, 2019. [Online; accessed 20-May-2019].

[10] Ladislav Čmolík and Jiří Bittner. Layout-aware optimization for interactive labeling of 3d models. *Computers & Graphics*, 34(4):378–387, 2010.

[11] Ladislav Cmolik and Jiri Bittner. Real-time external labeling of ghosted views. *IEEE transactions on visualization and computer graphics*, 2018.

[12] Franklin C Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH computer graphics*, volume 18, pages 207–212. ACM, 1984.

[13] Matt Escobar, 2017. [Online; accessed 20-May-2019].

[14] Herbert Freeman. Automated cartographic text placement. *Pattern Recognition Letters*, 26(3):287–297, 2005.

[15] Timo Götzelmann, Kamran Ali, Knut Hartmann, and Thomas Strothotte. Adaptive labeling for illustrations. In *Proceedings of Pacific Graphics*, volume 2005, pages 64–66, 2005.

[16] Timo Götzelmann, Kamran Ali, Knut Hartmann, and Thomas Strothotte. Form follows function: Aesthetic interactive labels. *Computational aesthetics*, 5, 2005.

[17] Timo Götzelmann, Knut Hartmann, and Thomas Strothotte. Agent-based annotation of interactive 3d visualizations. In *International Symposium on Smart Graphics*, pages 24–35. Springer, 2006.

[18] Knut Hartmann, Timo Götzelmann, Kamran Ali, and Thomas Strothotte. Metrics for functional and aesthetic label layouts. In *International Symposium on Smart Graphics*, pages 115–126. Springer, 2005.

[19] Justin Hensley, Thorsten Scheuermann, Greg Coombe, Montek Singh, and Anselmo Lastra. Fast summed-area table generation and its applications. In *Computer Graphics Forum*, volume 24, pages 547–555. Wiley Online Library, 2005.

[20] Mike Houston. Gpgpu benchmarking, 2007. [Online; accessed 19-May-2019].

[21] David Kouřil, Ladislav Čmolík, Barbora Kozlikova, Hslanc-Yun Wu, Graham Johnson, David S Goodsell, Arthur Olson, M Eduard Gröller, and Ivan Viola. Labels on levels: labeling of multi-scale multi-instance and crowded 3d biological environments. *IEEE transactions on visualization and computer graphics*, 25(1):977–986, 2019.

[22] Antonina Lebedeva. Web application collecting and evaluating data from user experiments, 2017.

[23] James R Lewis and Jeff Sauro. When 100% really isn't 100%: improving the accuracy of small-sample estimates of completion rates. *Journal of Usability studies*, 1(3):136–150, 2006.

[24] Michael Bekos Michael, Michael Kaufmann, Katerina Potika, and Antonios Symvonis. Polygon labelling of minimum leader length. 2006.

[25] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[26] Václav Pavlovec. External labeling of 3d objects, 2017.

[27] Renato Deris Prado and Alberto Barbosa Raposo. Real-time label visualization in massive cad models. In *2013 International Conference on Computer-Aided Design and Computer Graphics*, pages 337–344. IEEE, 2013.

[28] Guodong Rong and Tiow-Seng Tan. Jump flooding in gpu with applications to voronoi diagram and distance transform. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 109–116. ACM, 2006.

[29] Timo Ropinski, Jörg-Stefan Praßni, Jan Roters, and Klaus H Hinrichs. Internal labels as shape cues for medical illustration. In *VMV*, volume 7, pages 203–212. Citeseer, 2007.

[30] Jeff Sauro and James R Lewis. *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann, 2016.

[31] Thorsten Scheuermann and Justin Hensley. Efficient histogram generation using scattering on gpus. *SI3D*, 7:33–37, 2007.

[32] Thierry Stein and Xavier Décoret. Dynamic label placement for improved interactive exploration. In *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, pages 15–21. ACM, 2008.

[33] Edward R Tufte. *The visual display of quantitative information*, volume 2. Graphics press Cheshire, CT, 2001.

[34] Wikipedia contributors. Opengl — Wikipedia, the free encyclopedia, 2019. [Online; accessed 20-May-2019].

[35] Hsiang-Yun Wu, Shigeo Takahashi, Chun-Cheng Lin, and Hsu-Chun Yen. A zone-based approach for placing annotation labels on metro maps. In *International Symposium on Smart Graphics*, pages 91–102. Springer, 2011.

[36] Pinhas Yoeli. The logic of automated map lettering. *The Cartographic Journal*, 9(2):99–108, 1972.