3D Annotations for Geospatial Decision Support Systems

Silvana Zechmeister [*]
Supervised by: Jürgen Waser [†], Daniel Cornel [‡]

VRVis Research Center
Vienna / Austria

Abstract

In virtual 3D environments, it is easy to lose orientation while navigating or changing the view with zooming and panning operations. In the real world, annotated maps are an established tool to orient oneself in large and unknown environments. The use of annotations and landmarks in traditional maps can also be transferred to virtual environments. But occlusions by three-dimensional structures have to be taken into account as well as performance considerations for an interactive real-time application. Furthermore, annotations should be discreetly integrated into the existing 3D environment and not distract the viewer's attention from more important features. In this paper, we present an implementation of automatic annotations based on open data to improve the spatial orientation in the highly interactive and dynamic decision support system Visdom. We distinguish between line and area labels for object-specific labeling, which facilitates a direct association of the labels with their corresponding objects or regions. The final algorithm provides clearly visible and easily readable annotations with continuous levels of detail integrated into an interactive real-time application.

Keywords: Map Annotation, Geospatial Visualization, Open Data

1 Introduction

The economic growth and climate change have a great impact on the frequency and intensity of natural disasters worldwide. River floodings affect many people and cause a lot of damage and costs. Therefore, the need for flood management systems to analyze different flood scenarios grows, especially in densely populated areas with river proximity.
Visdom is a flood management system which supports interactive decision making based on fast geospatial simulation and visualization. It offers the opportunity to test different protection measures, to be prepared for flood disasters and to act correctly in serious situations. A good support of spatial orientation and navigation in urban 3D environments for flood managers, relief workers and other persons involved is needed. A common approach to achieve this aim is the use of annotations. Thus, this paper covers the extension of Visdom with labels for different landmarks to know where the affected areas are located and to stay oriented while navigating through the system.

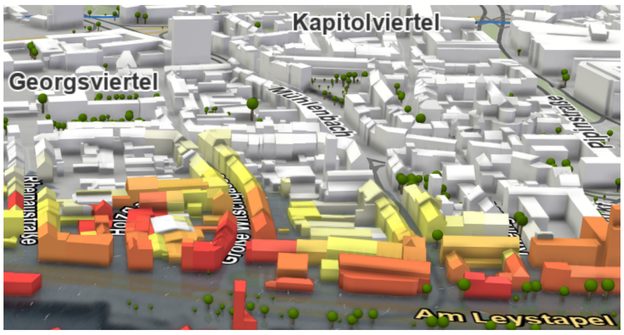

Figure 1: Area and street labels integrated into the 3D environment without anchor elements.

The integration of annotations in a 3D dynamic environment causes different challenges compared with static 2D city maps. In contrast to static approaches, dynamic systems need a good trade-off between computation time and optimal label placement to enable a real-time user interaction. In Visdom, the annotations should not impede the user by visual distraction or by hiding important simulation data. The major goal is not to place as many labels as possible but to support the navigation in a subtle way. These requirements increase the importance of a good visual integration into the dynamic 3D environment without additional anchor elements (see Figure 1).
The label data used for the annotations are from Geofabrik [4], a free download server which extracts geospatial data from OpenStreetMap [10] and provides map textures, line and area shape files. The textures do not allow the partitioning of label data and thus updating comes along with loading high volumes of data and long waiting times. This impedes dynamic updates during runtime, while the label resolution, orientation and size have to stay static. Their inflexibility is not applicable with the high interactivity and large zoom range of Visdom. To guarantee label readability, their font size and orientation should dynamically adapt to the zoom factor and the point of view. Furthermore, texture based labels mapped to the ground are

[*]zechmeister@vrvis.at
[†]jwaser@vrvis.at
[‡]cornel@vrvis.at

unsuitable for annotating 3D objects such as buildings because they are hiding their own labels. OpenStreetMap offers an extensive amount of data which contains line and area segments with their own labels and partially incorrect label text (e.g. "NULL"). The high segmentation of lines results in too many labels in the most cases. To produce appropriate annotations, preprocessing and intelligent label placement has to be executed. When placing a label along a line, the positioning of its letters is not trivial. They have to follow arbitrary curves which can have a negative impact on label readability. Thus, the aim is a label appearance, which is as straight as possible. To make the depth information more assessable, the annotations should be occluded by other scene objects. The overlap between two labels should be avoided, since this would destroy their legibility. It is essential in a real-time application such as Visdom, to efficiently test labels for intersection with others.
This paper presents an implementation of annotations which solves the mentioned problems and eases the localization and identification of streets, rivers, buildings, places, city districts and other landmarks.

2 Related Work

Prior research dealt with automating and optimizing the annotation process concerning the quality and quantity of labels placed on static 2D maps. Research in the area of dynamic 3D environments with pan and zoom also tries to accelerate the execution time of the label placement process. There is a wide area of application for labeling features, it is used in cartography, geographic information systems and point pattern analysis for instance. Thus, there exist various techniques to find a label placement and layout according to the annotation application.
Marks and Shieber [9] showed that label placement on a map is an NP-hard problem and it needs heuristics to label large quantities of features. There are force-based [3], slider-based [12] and other algorithms to find an appropriate label placement. To find best possible label positions, the definition of the optimum is essential. Van Dijk [2] reviewed existing label placement rules and defined four categories to evaluate label placement quality. The most common label features are points, lines and areas to annotate cities, streets and countries for instance. Depending on the kind of feature, different issues have to be overcome to determine good label positions. The four feature dependent quality criteria of Van Dijk are aesthetics, label visibility, feature visibility and label-feature association.
The use of signed distance fields is a way of text rendering which tries to overcome the limitation of raster graphics [5]. The signed distance field texture can be very small and it still produces crisp text. But this approach does not accurately represent text contours near to complex intersections and it is not appropriate for very small text sizes [6]. In such cases, the use of bitmap graphics or the direct rasterization of vector graphics may produce better results. Depending on its appearance, a text can attract attention, be seamlessly integrated into the environment or increase legibility and the appealing look of the whole 3D scene. Adding an outline to letters can increase their contrast to the background. The use of halos around labels, which clear the space close to them, leads to further readability improvement [14]. Vaaraniemi et al. [13] introduce methods to enhance the visibility of labels which are occluded by other objects of the 3D environment. They propose a transparency label aura and glowing streets for street labels to achieve this goal. The dynamic change of label position and orientation enables the adaption to user interaction for better visibility and readability. But such behavior can result in flickering and thus distract the user. To avoid this effect, Maass and Döllner use label blending and animation [8]. With blending, the labels smoothly fade in before appearing and fade out before disappearing. The animation is used to continuously move a label from one position to another.

3 Preprocessing

The first step of the annotation pipeline is the preprocessing of input data provided by OpenStreetMap [10]. The data need to be prepared to achieve an efficient further use and to enable fast rendering. The incoming data are labels and two different shape types. There are optional importance and color values available to adapt the label output accordingly. A label is assigned to a certain shape and a position on or near this shape. Its corresponding text might be empty or "NULL", depending on the OpenStreetMap data. In this case the label is not taken into account. The two different input shapes are lines and 2D polygons, which are used for streets, places, buildings and other landmarks. A line is defined by a number of control points. The connection of all points results in the line to which the control points belong. A polygon is defined by a set of consecutive vertices, whereby the first and the last vertex are the same.
The OpenStreetMap data include numerous street lines and each represents a street segment with its own label. If we would render all these labels, it would result in an overloaded and confusing scene. To avoid this, we want to merge lines and their corresponding labels together. Our approach is to merge two street segments together if they belong to the same street and are adjacent to each other. When merging street segments, the labels assigned to them are merged too. This is done by setting the merged label into the middle of the new merged line segment. But some restrictions are needed for the line merge process to prevent merging street segments when the streets cannot be clearly identified. This is the case if it is not obvious which direction a street course is following. To avoid such a scenario we check the angle between the potential end positions to be merged and if it is acute-angled, we do not

merge. This results in streets, which are labeled before and after strong bends. The polygons have at most one label, so there is no need for a reduction as for the lines.
The next preprocessing step after the line merge is the label sorting. Labels are sorted according to their camera distance and optional importance values. The importance values are used to improve the final output. More important labels are visually differentiated compared with less important labels and get a preferential treatment. More relevant labels are rendered first to increase the probability of their visibility. But independent of the importance values, labels near the camera should be paid particular attention, since these are most likely labels the viewer is interested in. Thus, labels with the same importance are treated according to their distance to the camera. The camera distance changes more frequently than the importance values because it depends on the navigation through the environment. The labels have to be sorted by their camera distance each frame, but their order according to their importance stays the same and can be precomputed during the preprocessing.
Since there is no native support for text rendering in OpenGL, a custom implementation is necessary. For the application of rendering label text in Visdom we decided to use vector graphic fonts which allow an adaptive rasterization for all required font sizes. In the field of typography, characters are represented by glyphs and a vector graphic font contains accessible glyph metrics. With a given label position, each letter can be placed appropriately by the use of its metrics. All labels are rendered with the same font, only in different size and therefore its glyph metrics can be loaded and stored for later use. Furthermore, texture atlases containing the most common 256 letters and their outlines are generated through rasterization of the letter's vector graphic descriptions. There are several texture atlases needed to cover all required font sizes. If only one texture atlas would be used and scaled to fit the appropriate font size, the letters would have a poor, upscaled resolution. Thus, texture atlases for all initial font sizes are generated in the preprocessing step.
The last preprocessing step concerns the label orientation and its great impact on readability. Most languages have a write direction from left to right and therefore one is trained to read in the same direction. This holds true for the annotations in Visdom, which are mostly in German. This is why we aim to orient the labels from left to right along the street segments to ease reading. The order of line control points determines the orientation of the corresponding street label. To adapt the label orientation appropriately, the control point order needs to be dynamically changed according to the current view. For a quick access during rendering, the original and the reversed order of control points are stored.
With this prepared and quickly accessible information, text rendering can be executed in a fast and efficient way.

4 Label & Letter Placement

After processing the input data, the labels and their individual letters need to be placed at the right position. The label placement is a very complex task with several possible solutions. Some of them are already mentioned in section 2. The main criterion for label and letter placement in Visdom is the execution time. It has to be very fast to be able to afford real-time interaction. But a good placement has to adapt to the current view to improve its readability by changing its scale and orientation. As a result, label placement is view-dependent and is therefore changing very often and needs plenty of time to process. Due to this preconditions, the aim is not to find an optimal label placement but a satisfactory and fast placement. The use of fixed instead of dynamic label positions turned out as a good choice. It avoids flickering and saves important runtime through skipping dynamic calculations of label positions. After the determination of a street or area label's position, each character has to be placed along a street or above a point of interest. Contrary to the determination of the label position, the letter placement is executed dynamically to change its orientation and scale according the current view. By means of the glyph metrics, the letters can be placed in the scene with their positions only. Thus, the aim is to get the positions for all letters.

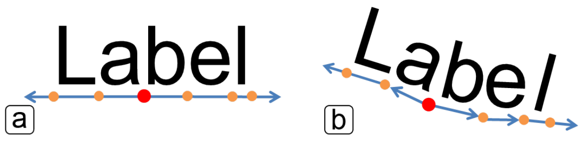

Figure 2: Illustration of label (red) and letter positions (orange) on a horizontally placed area label (a) and a street label placed along its corresponding street (b).

4.1 Area Labels

To determine the letter positions it is necessary to get the floating label positions first. This is accomplished by shifting up the area's midpoint. The label position (see Figure 2a, red) is then the position of the middle letter too. Starting from this point each letter is placed horizontally to the screen by using its corresponding advance. The glyph advance is the horizontal space needed to place a certain letter accurately and can be extracted from the glyph metrics. The letter positions are calculated by subtracting the glyph advance if the letter is to the left of the label position and adding it otherwise. The orange points of Figure 2a represent the positions of the letters and the space between them is referred to as the advance. The final output is a label which is horizontally centered and floating above its corresponding area. Caused by using the screen-space for the placement, area labels have a constant size and are always facing the viewer. This means they have a high readability because the text is not scaled when zooming in or out, has no distortion and is always orthogonal to the screen.

4.2 Street Labels

In consequence of fixed label positions, every street label is permanently located at the middle of its street segment. A usual practice of street label placement is the projection of the labels onto the streets to increase the label-feature association. But to achieve this, the letter placement needs to be performed in world-space. The constant size of area labels turns out to be handy because the labels are never over- or under-dimensioned and are always easily readable. To get such a constant label size also for world-space placement, a scale factor is used, to resize letters according the current zoom factor. The field of view and the distance between the label and the camera is taken into account to get an appropriate scale factor. This factor enables a nearly constant label size with respect to the screen and thus facilitates better label readability. The letter placement starts at the label position, which is the red point in Figure 2b. Then moving along the line by the same glyph advance procedure as for the area labels. During the letter placement, one additional process of improvement is done. The averaging of letter positions reduces heavily bended labels which creates a more connected label appearance and a smoother label course. The letters are averaged by setting the letter positions to the center of their predecessor and successor. After this step, the spaces between these letter positions do not comply with the associated glyph advances. To correct them, the positions are shifted towards their neighbor position accordingly. This averaging procedure is done for all letters with two neighbors. Since heavily bended labels are hardly readable and visually less appealing, a label is not rendered if the angle between its averaged letter positions exceeds a certain maximum.
The determined label and letter positions facilitate the execution of visibility tests in the next step.

5 Visibility Tests

In the previous step, labels and their letters are assigned to a certain position. If all of them were rendered, the result would be an overloaded scene with worse label readability and labels covering important information. To avoid this scenario, intersection tests are implemented to determine if a label is overlapping with others. Since the labels are rendered by descending importance and ascending distance to camera, only a less or equally important label could occlude another label. When a label overlaps with another, it is not rendered. If the importance values are equal, the distance to the camera plays the decisive role. For more efficiency, the intersection tests are split into three stages. The first stage represents a search grid, which enables intersection tests only with labels in the same region. These regions are cells in a grid covering the entire domain. Only labels which are lying in the same cell as the current label are possible intersection candidates. The cell access is easy and fast and the search grid avoids a lot of unnecessary intersection tests.
The second and third stages are the intersection test itself with two different kinds of bounding boxes which differ in their level of detail. After the first step, all possible intersection candidates need to be tested for overlap with the current label. At first, this is performed with axis-aligned label bounding boxes. These bounding boxes enclose all letters of a label and are only defined by four vertices. This makes them simple and fast, but inaccurate. The area covered by this kind of bounding box may be much larger than the label really needs, since it is always an axis-aligned rectangle. By only using the label bounding boxes, there would be many false-positive intersections. Therefore, we use a second kind of bounding box for intersection testing, which is only performed if the first bounding boxes are intersecting. The individual letter bounding boxes are used to test for overlap of single letters. This is more precise, but needs more execution time because it has to be performed for each letter. If this last intersection test detects an intersection, the current label is not rendered.

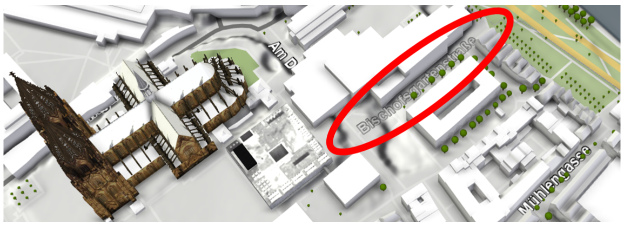

Figure 3: Fading out the label "Bischofsgartenstraße" because it is running out of space.

The abrupt change of label status from visible to invisible and vice versa triggers label flickering. Therefore, a smooth fade-in or fade-out is performed by lowering or increasing label transparency on certain cases. It is possible to predict if a street label is running out of space on a street segment because it may be located at the boundaries of its street segment. Then the transparency is gradually reduced according to the remaining space. In Figure 3, one can see the label "Bischofsgartenstraße" fading out, caused by the short space remaining of its corresponding street segment. We already discussed that labels which are far away are less interesting for the viewer than near labels. Due to this fact, we aim to make labels far away disappear. This increases the possibility that labels which belong to streets, places and other landmarks near the viewer are visible. To also avoid flickering in this case, they are faded in and out like labels which are running out of space.
Until now, we have only discussed intersections between labels, but a usual scene in Visdom includes many different objects like buildings, protection lines, trees and mountains, which also influence the label visibility of street labels. Area labels are rendered floating above everything else, therefore we do not need to consider occlusion by

other objects. It is difficult to deliver a realistic depth perception and make street labels visible through other objects at the same time. For perspective views, we decided to preserve the depth information for close street labels and give it up on growing distance. Labels have constant size with respect to the screen, but other objects have constant size with respect to the 3D world and they are getting very small if they are far away. Then their detailed visual information is no longer detectable. Therefore, it is less irritating that the labels are visible through occluding objects if they are far away. In the case of orthographic views, the depth of buildings and other objects, which may hide street labels, is hardly perceptible by the user. Thus, the annotations are always visible through them without destroying the 3D perception. The visibility tests determine the labels passed to the last step of the annotation process to finally render them.

6 Rendering

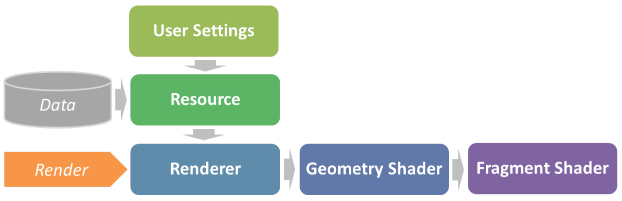

Figure 4: Overview of internal structure and information flow (gray arrows) of the render process.

The last step of the annotation process is the rendering of all labels which passed the visibility tests. The orange arrow in Figure 4 visualizes the invocation of the *Renderer* class by the logic of Visdom to start a new render pass in each frame. When the *Renderer* becomes active, it starts the annotation render process with the data received by the *Resource* class which manages the label data. These data have different sources and attributes. The data delivered by OpenStreetMap [10] are constant at run-time and the glyph metrics extracted from a given font do not change either. Therefore these data represent the static part of the data managed by the *Resource*. The user settings represent the dynamic data because the values can be modified by the user during execution. If the user changes them, the *Resource* updates all dependent resources and provides the refreshed data for the *Renderer*. If there are no changes made by the user, the default font is the Google font Roboto [11], which has a special design to make label text more legible even with a small point size. To increase the contrast between labels and their background and to further improve their readability, outlines are added to the letters. The size and color information of a label indicate the label relevance for the user. The outline color is chosen as white or black, depending on which color gives the highest monochrome contrast to the font color. The default font is bold to balance the ratio between the label letters and their outline and to improve annotation text legibility.

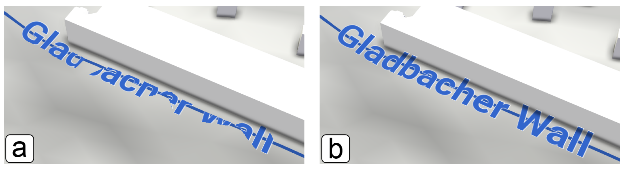

Figure 5: Occlusion issue with standard depth test (a) and desired behavior with modified depth test (b).

According the current view, the *Renderer* uses the updated resource data for the label and letter placement and for the execution of visibility tests. After these steps, it passes label texts, letter positions, scale factors and the glyph metrics to the geometry shader to span the quad for every letter. This is done on the GPU to save important runtime. At this point, every character has only one position and the geometry shader uses the glyph metrics to calculate four vertices out of this one. But since there is only one real point in the 3D world of Visdom, there is also only one depth value. For area labels this does not matter, since they do not need a precise depth value because they are not depth tested with the rest of the scene. This is due to the fact that area labels are floating above everything else. But for street labels, the missing accuracy of depth values is a real problem. This one depth value per character lies directly on the street line and if the current view plane is not exactly parallel to the line, the four generated vertices out of this one appear wrong and are sometimes occluded by the terrain (see Figure 5a). To avoid such depth errors, the OpenGL integrated depth testing is disabled and an own depth test is performed in the fragment shader. The result of the own depth test implementation is shown in Figure 5b. This test uses a depth offset which complies with the letter height which is the maximum depth error. If the letter depth with this offset is closer than the rest of the 3D environment, the letter will be drawn. The *Renderer* passes color information and texture atlases of the characters and their outlines to the fragment shader. Afterwards, the fragment shader is able to finally draw the label characters to the global frame buffer of the Visdom scene.

7 Results

The described annotation process of the last sections is implemented with the use of the graphics API OpenGL and the programming language C++. All fonts of the annotations in Visdom are of the font format TrueType. It is a common font format which stores font information by the use of quadratic Bézier splines, which classifies it to the group of vector graphic font formats. To access and handle the font data provided by TrueType, the library FreeType [7] is used. It eases the extraction of glyph metrics and the

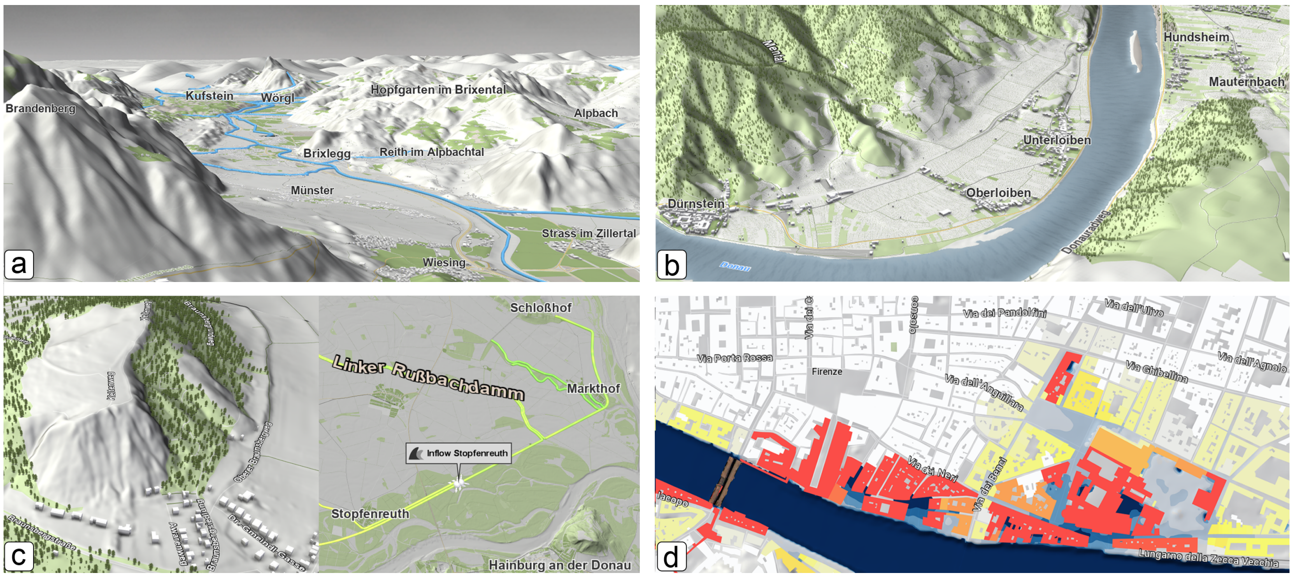

Figure 6: The first case study (a) shows the annotations used in the mountainous environment of Tyrol. In Wachau (b) one can see labels in narrow valleys and used for the Danube river. The Marchfeld case study (c) shows labels nestling up to hills and used for dikes. The last case study (d) depicts an orthographic view of Florence with Italian labels.

rasterization of vector graphics to get appropriate bitmap font data. If the used font does not provide hinting information, FreeType offers auto-hinting to make text more legible. But missing kerning information cannot be replaced and is therefore a strict requirement for the chosen font.

We use different case studies to analyze the visual appearance and behavior of the annotations in Visdom. There are five flood management case studies, three of them are located in Austria, one in Germany and one in Italy. They differ in attributes such as landscape, amount and type of label data and in the case of Italy, in language of annotations. The case studies do not only consist of densely populated cities but some also cover a wider range with multiple cities and villages. Some also have hills, mountains, rivers and dikes, where the annotations are applied like in cities. These varied case studies offer the opportunity to see how the annotations act when the environment and the provided label data are changing. They ease the analysis of the overall strengths and weaknesses of the annotations. For this purpose, performance tests are executed in the five case studies in different scenarios. These scenarios cover different views, from orthographic to perspective side views and from high to low zoom levels.

7.1 Case studies

The fist case study is Cologne, in which the annotations where initially implemented. There is a high amount of label data to process which is a challenge for their dynamic real-time rendering. The terrain is relatively even with only very low height. This fact improves the appearance of street labels, because their individual letters are

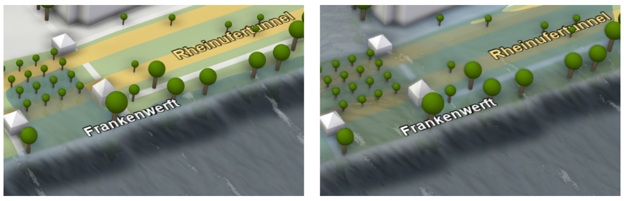

Figure 7: Street labels dynamically adapt to the water surface of the flood simulation to stay afloat.

not changing orientation heavily when adapting to the underlying ground. In the Cologne case study, the annotations are not only used for streets, buildings, places and landmarks, but they are also applied to labeling protection barriers and sewer networks. In Figure 7 a small part of Cologne is shown. One can see labels rising with the flood, which enables their legibility during the entire flood simulation, even in inundated areas.

The case study of HORA (= Natural Hazard Overview & Risk Assessment Austria [1]) sticks out with the most mountains and rivers in the valleys. The area of Tyrol shown in Figure 6a is characterized by the mountains of the Alps. The rivers are emphasized with light blue lines for better river visibility during flooding. One can see that the floating area labels are well-suited for city and village labels in mountainous landscapes. But when labeling rivers and streets on mountain slopes or sharp ridges, the labels may be heavily bended and thus hardly readable or disappear when reaching the maximum bend angle.

The use of the annotations in narrow valleys or on hills can be observed in the case study of Wachau (see Figure 6b). In addition, the Figure shows the annotation of the large Danube river along its center line as well as the combina-

tion of labeling flat areas in the valley and uneven ground on the hills. Compared to the HORA case study, the labels are disappearing less often since the hills of Wachau do not have such steep slopes as the mountains of Tyrol.
The dynamic adaption of the label orientation to make text always readable from left to right is still correct on borderline cases such as the almost vertical labels in Figure 6c. The labels are still easy to read even if they nestle up to gentle hills and bend along curvy streets. The Marchfeld case study uses annotations also to label dikes.
The last of the five introduced case studies is Florence. The annotations behave similarly to Cologne because they are both cities with relatively even ground. Florence has narrower streets and is populated more densely than Cologne. Thus, there are more streets over the same area and therefore more labels to process. Figure 6d shows the labels of Florence in orthographic view where the modified depth test shows its positive effect. Using the standard depth test would result in partially hidden street labels behind buildings. One can also see that the color-coded buildings and terrain attract attention but the labels are still clearly visible and legible.
An effect which is visible in all five case studies is that the annotations are changing their level of detail according to the zoom factor. This property is a positive side effect of the implemented visibility tests and consideration of label importance. When zooming out, labels which are less important are disappearing and give way to more important labels. By zooming out, more labels are overlapping with each other and only the labels with higher relevance have the privilege to be rendered. When zooming in, fewer labels are overlapping and less important labels get enough space and become visible.

7.2 Performance Tests

In this section, the results of performance tests on the introduced case studies are presented. The tests were executed on a computer with 32 GB RAM and Intel Core i5-4690K CPU with 4x 3.50 GHz and with a Nvidia Geforce GTX 980 graphic card with 4 GB. The tests are separated into the two categories update and render duration. The update process is executed only on the CPU. By counting CPU cycles, the elapsed time is measured. For the update tests, only the initial update process involving the entire data set is considered because later updates are only processing subsets of the data. The render duration takes into account the time needed for the placement, the visibility tests and the rendering itself. These processes are performed on both CPU and GPU. The elapsed GPU time is measured with a pipeline statistics query exposed by the OpenGL API.

The distribution of shapes over all case studies is visualized in Figure 8 (right). Wachau is the only case study which has more polygons than lines. In all other case studies, line data dominate, which require more line merges during the initial update process. The number of lines and

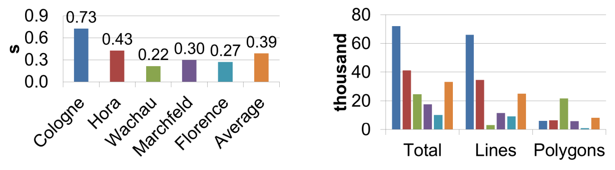

Figure 8: The time needed to update all label data (left) and the distribution of shapes (right), both per case study.

the update time are strongly correlated. For the case study of Cologne, OpenStreetMap provides the most label and shape data of all case studies. In Figure 8 one can see, that the initial preprocessing of all label data of the largest case study with over 70,000 shapes (right) is faster than one second (left).

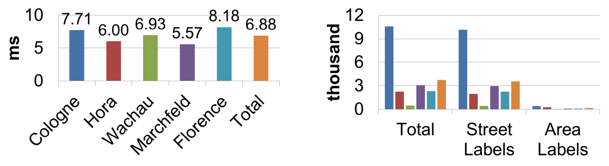

Figure 9: The average time needed for one render pass per case study (left). The number of street and area labels and the total number of labels per case study (right).

The average time needed to render these labels is visualized in Figure 9 (left). The number of labels rendered lies between 20 and 30 labels on average. Since the most labels are rendered in the case study of Florence and the least in Marchfeld, they have the longest and shortest average render duration. In Figure 9 (right) the number of labels after the update process is shown. Compared with 8 (right) there are far less labels than shapes because labels with invalid text are discarded and street labels merged. Focusing on the Cologne case study, the processing of over 10,000 labels and the rendering of labels passing the visibility tests took 7.71 ms. These results show that all case studies can be rendered at highly interactive frame rates and that the annotations allow fluent dynamic real-time interaction with Visdom.

8 Summary & Future Work

This paper covers the implementation of annotations for an interactive 3D application. Two different label approaches for line and area labels facilitate an object-specific labeling, which enables a direct label-feature association without additional objects. The dynamic adaption of the label orientation to user interaction and changing views increases the label legibility. Through the implemented occlusion handling, the depth perception in close-up views is preserved and in far-away views the labels are still readable. The annotations provide continuous levels of detail, whereby the label visibility depends on the label importance and the zoom level. The scaling according to the

zoom level fixates the label size relative to the screen and preserves label legibility. The implementation supports processing large quantities of labels in real time and with a fluent user interaction. The use of fixed label positions speeds up the placement process because only the individual letters need to be placed dynamically and not the whole label. The preparation of all data which are not changing permanently is also saving important runtime. The three step visibility tests with the search grid and label and letter bounding boxes represent a further performance improvement. Finally, the GPU-side vertex calculation of the label letters enables the use of only one position per letter for CPU processing which also contributes to the high interactivity. Thus, the implementation fulfills all measurable initial requirements.
The different case studies demonstrate that the annotations are applicable to different scenarios. But there are some known issues. The street line merge process does not reliably detect all crossings and junctions to provide an optimal label-street association. The problem of one depth value per letter causes imprecise depth tests with the environment which can lead to labels partially sinking into the ground or labels visible in front of objects which should cover them. This behavior is only appearing in very special cases such as a highly uneven terrain. As a result of the fixed label positions, the labels are often truncated or not visible even if their feature is. This is noticeable especially in the case of long streets with only one label. We consider a more flexible, but temporally stable label placement for very long streets an important direction for future work.

9 Acknowledgments

I would like to thank my supervisors Jürgen Waser and Daniel Cornel for the good cooperation, the great support and their patience. This work was enabled by the Competence Centre VRVis. VRVis is funded by BMVIT, BMWFW, Styria, SFG and Vienna Business Agency in the scope of COMET – Competence Centers for Excellent Technologies (854174) which is managed by FFG.

References

[1] Bundesministerium für Nachhaltigkeit und Turismus. Natural Hazard Overview & Risk Assessment Austria. http://www.hora.gv.at, 2018. Accessed: 2018-01-15.

[2] Steven Van Dijk, Marc Van Kreveld, Tycho Strijk, and Alexander Wolff. Towards an evaluation of quality for names placement methods. *International Journal of Geographical Information Science*, 16(7):641–661, 2002.

[3] Dietmar Ebner, Gunnar W. Klau, and René Weiskircher. Force-Based Label Number Maximation. Technical report, Vienna University of Technology, Institute of Computer Graphics and Algorithms, 2003.

[4] Geofabrik GmbH. Maps & Data. http://www.geofabrik.de/data, 2017. Accessed: 2017-10-27.

[5] Chris Green. Improved Alpha-tested Magnification for Vector Textures and Special Effects. In *ACM SIGGRAPH 2007 courses*, pages 9–18, 2007.

[6] Stefan Gustavson. 2D Shape Rendering by Distance Fields. In *OpenGL Insights: OpenGL, OpenGL ES, and WebGL community experiences*, pages 173–182. CRC Press, 2012.

[7] Werner Lemberg. FreeType Overview. https://www.freetype.org/freetype2/docs, 2017. Accessed: 2017-11-27.

[8] Stefan Maass and Jürgen Döllner. Embedded Labels for Line Features in Interactive 3D Virtual Environments. In *Proceedings of the 5th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa*, pages 53–59, 2007.

[9] Joe Marks and Stuart Shieber. The Computational Complexity of Cartographic Label Placement. Technical report, Harvard Computer Science Group, 1991.

[10] OpenStreetMap Contributors. OpenStreetMap. https://www.openstreetmap.org, 2017. Accessed: 2017-10-23.

[11] Christian Robertson. Google Fonts Roboto. https://fonts.google.com/specimen/Roboto, 2018. Accessed: 2018-01-15.

[12] Nadine Schwartges, Jan-Henrik Haunert, Alexander Wolff, and Dennis Zwiebler. Point Labeling with Sliding Labels in Interactive Maps. In *Connecting a Digital Europe Through Location and Place*, pages 295–310. Springer International Publishing, 2014.

[13] Mikael Vaaraniemi, Martin Freidank, and Rüdiger Westermann. Enhancing the Visibility of Labels in 3D Navigation Maps. In *Progress and New Trends in 3D Geoinformation Sciences*, pages 23–40. Springer Berlin Heidelberg, 2013.

[14] Mikael Vaaraniemi, Marc Treib, and Rüdiger Westermann. Temporally Coherent Real-time Labeling of Dynamic Scenes. In *Proceedings of the 3rd International Conference on Computing for Geospatial Research and Applications*, pages 17:1–17:10, 2012.