

Silhouette-Based Label Placement in Interactive 3D Maps

Christine Lehmann and Jürgen Döllner

Abstract This paper presents a silhouette-based technique for automated, dynamic label placement for objects of 2D and 3D maps. The technique uses visibility detection and analysis to localise unobstructed areas and silhouettes of labeled objects in the viewplane. For each labeled object, visible silhouette points are computed and approximated as a 2D polygon; the associated label is finally rotated and placed along an edge of the polygon in a way that sufficient text legibility is maintained. The technique reduces occlusions of geospatial information and map elements caused by labels, while labels are placed close to labeled objects to avoid time-consuming matching between legend and map view. It ensures full text legibility and unambiguity of label assignments by using actually visible 2D silhouette of objects for label placement. We demonstrate the applicability of our approach by examples of 3D map label placement.

Keywords Labeling · Text visualization · Information visualization

1 Introduction and Related Work

Label placement or labeling is an old discipline that has its origin in 2D cartography (Imhof 1985). Different labeling strategies have been devised to attach textual information to objects. A large number of labeling strategies exist for point feature labeling of 2D objects (Bekos et al. 2010; Christensen et al. 1994; Mote 2007; Wagner et al. 2001) and 3D objects (Stein and Décoret 2008; Maaß and Döllner 2006b). The challenge in point feature labeling comes from the high density of labeled objects or Point-of-Interests (Kern and Brewer 2008).

C. Lehmann (✉) · J. Döllner
Computer Graphics Systems, Hasso-Plattner-Institute, University of Potsdam,
Potsdam, Germany
e-mail: christine.lehmann@hpi.uni-potsdam.de

Algorithms commonly use external lines to connect label and labeled object, which is called *external labeling*. Fewer strategies exist for line feature labeling (Maaß and Döllner 2007) and area feature labeling (Maaß and Döllner 2006a), which overlay objects with labels. Such approaches are referred to as *internal labeling* approaches.

From cartography, general principles and requirements on text labels are well-known (Imhof 1985), including *legibility*, *clear graphic association*, and, in particular, *minimum covering* of map content. In particular, the last issue represents a major factor that affects the clarity of the visualization (Christensen et al. 1994). With internal labeling, much content is covered by labels. With external labeling, only one point of the labeled object is required. In case of illustrative labeling examples, i.e., labels can be placed exclusively around the content and thus provide less covering of labeled objects. However, for general labeling examples, external lines can also cover adjacent labeled objects.

In contrast to *static* label placement, *dynamic* label placement requires that labeling algorithms are suitable for responsive visualization systems and interactive applications (Kopetz 1993) and thus perform all computations fast enough. Furthermore, interactive applications are characterized by frequent changes in view and—in case of 3D visualizations—also by changes in perspective. This makes it more difficult for the user to focus and perceive the desired information, compared to a static map, for example, where the user has more time to study the given single view. In the field of human user interfaces, this refers to the task of the user to keep the “locus of attention” (Raskin 2000) in interactive applications, i.e., to detect and interpret the difference between two consecutive views of the visualization. Hence, if there is too much information visualized, the user loses the locus of attention and consequently does not perceive the message of the visualization anymore. As a conclusion, our labeling method focuses on providing an appropriate amount of information to the user using visibility-driven label placement (Lehmann and Döllner 2012). Using visibility information of labeled objects, which we call *visibility-driven* label placement, is essential for dynamic label placement, as it reduces missing labels and label cluttering (Lehmann and Döllner 2012).

In this paper, we present a novel labeling strategy, called *silhouette-based label placement* that provides high visibility of underlying map content without losing spatial closeness between label and labeled object. Labels are placed along the visible silhouette of labeled objects (Fig. 1). In this sense, silhouette-based labeling is a balance between external and internal labeling. Our method is fully automated and does not require manual refinement. The label placement is renewed for each frame and reacts on interactive frame rates. Hence, it is suitable for interactive visualization applications.

We will further present how text and character transformations, which are necessary to align labels to visible silhouettes, can be efficiently implemented. With our font rendering method, we overcome legibility problems caused by aliasing artifacts and enable dynamic transformations of text and single characters.



Fig. 1 Occlusions of labeled objects caused by labels are reduced to preserve their information, while fast label assignment is enabled

Our GPU-based rendering method for text labels can be easily integrated into existing rendering applications, e.g., into server-side applications that need text visualizations, as the main part of text rendering is placed in a pixel shader. We demonstrate the applicability of our labeling method for 2D and 3D visualization applications by city and indoor labeling examples.

2 Silhouette-Based Label Placement

For each labeled object, pixels of the visible area are detected. As this detection is performed in the image space, where both 2D and 3D objects are represented by 2D regions, our method can be applied for 2D and 3D maps. Visible regions of 3D objects can consist of separate parts, for instance caused by occluding foreground objects. These parts or clusters in the visible point set are detected using flat Euclidean clustering (Jain et al. 1999). The cluster containing the highest number of points is selected for further processing (Fig. 2). From the cluster, we extract the boundary points or *the visible silhouette*. For each boundary, the vector pointing to the next inner point is computed, which we call the *inner vector* in this context. Hence, we call the resulting set of inner vectors for a given point set the *inner vector field*. In the following, we present our completely automated technique for aligning labels to visible edges of arbitrary 2D and 3D objects.

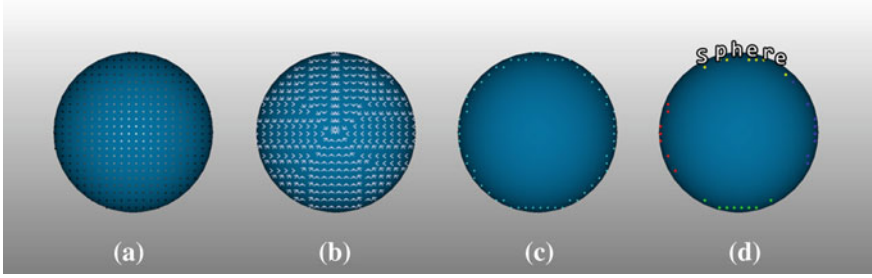


Fig. 2 Visible regions of labeled objects (example: spherical object) are analyzed in image space. **a** The *distance field* describes the distance for each visible point to the boundary. Points with highest distance are depicted white. Using the distance field, the visible silhouette (**c**) is selected. **b** The *inner vector field* describes the direction from each point to the inner points. It is used to detect *edges* in the visible silhouette (**d**), i.e., *left edge* (red), *bottom edge* (green), *right edge* (blue), and *top edge* (yellow). In this example, the label “Sphere” is aligned to the *top edge*

2.1 Detecting Visible Edges

We define an *edge* as a sorted list of 2D points that describes one side of the visible boundary of a 3D object. Thereby, the edge points are sorted in ascending order by their x -value. From the visible silhouette, i.e. the visible boundary points, we classify into top, bottom, left, and right edge (*edge classes*). Visible regions can have various shapes, for example, a shape closely matching a circle or a rectangle. The challenge is to find edges for arbitrary shapes of visible regions. This is difficult in case of shapes closely matching a circle, as a circle naturally has no edges or sides. However, we can analyze a boundary point relatively to the inner points of the shape. Hence, we use the inner vector field of the shape for edge detection.

To assign a point P_i to the correct edge class, the inner vector field is used: the point’s inner vector N_i points to the inside of the visible region. Hence, we distinguish edges using the dot products $\langle N_i, E_0 \rangle$ and $\langle N_i, E_1 \rangle$, with $E_1 = (-1, 1)^T$ and $E_0 = (1, 1)^T$ (Fig. 3). For edge classification, only the sign of the dot product is relevant, thus, normalization of E_0 and E_1 is not required. Visible edges are classified using the following equations:

Bottom edge: $\langle N_i, E_0 \rangle > 0$ and $\langle N_i, E_1 \rangle > 0$.

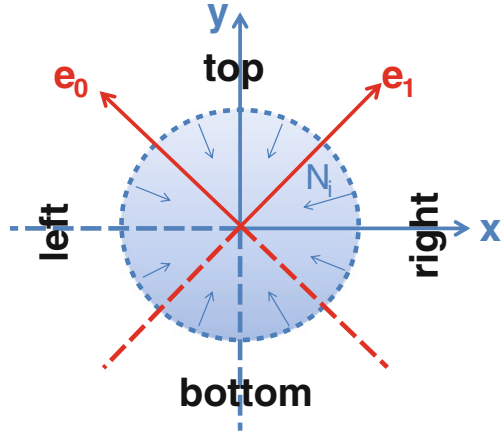
Top edge: $\langle N_i, E_0 \rangle < 0$ and $\langle N_i, E_1 \rangle < 0$.

Left edge: $\langle N_i, E_0 \rangle > 0$ and $\langle N_i, E_1 \rangle < 0$.

Right edge: $\langle N_i, E_0 \rangle < 0$ and $\langle N_i, E_1 \rangle > 0$.

There are several ways to understand *bottom*, *top*, *left*, and *right* depending on the reference coordinate system. We can use the coordinate system defined by the eigenvectors of the visible point set, i.e., the 2D object or local coordinate system, or the axis-aligned coordinate system, i.e., the image space coordinate system. We

Fig. 3 To detect edges in the visible silhouette of labeled objects, the diagonal vectors e_1 and e_0 are used, i.e., the coordinate axes x and y rotated by 45° . Using the normal field and the diagonal vectors, points are assigned to one of the four regions *left*, *right*, *top*, and *bottom*



use the axes of the image space coordinate system to compute the absolute top, bottom, left, and right edge. When using a local coordinate system, the determination of common edges between adjacent labels is difficult, as edges cannot be compared without complex coordinate system transformation.

In case of concave shapes or shapes with holes, multiple edges can be detected for one edge class. In this case, the edges must be separated using flat clustering by the Euclidean Distance, i.e., the pixel distance between the edge points.

2.2 Selecting Edges

To select an edge, to which a given text can be aligned, we apply the following criteria:

1. The edge length in pixels must be sufficient for the given text length in pixels.
2. Horizontal edges (top and bottom edge) are preferred over vertical edges (left and right edge).
3. The last selected edge is preferred, if it has a sufficient length, so that spatial coherence between two consecutive views is improved.

Regarding criterion 1, the font size can be decreased to fit the text into the longest edge, if no edge is sufficiently sized for the text in original font size. According to cartography principles, objects of the same or similar type should obtain text labels with the same or similar font. Consequently, we assume that font size is constant for all labels, and blend out labels that do not fit to one of the four edges.

Regarding criterion 2, we set the following order for edge priorities in our examples: the highest priority is assigned to the top edge, followed by bottom edge, left edge, and right edge. The choice of edge priorities is related to reading

direction and, thus, depends on cultural aspects in text processing. For this reason, priorities are user-defined in our implementation.

Labels are placed along the boundary, but still inside the visible area of the labeled object. Hence, occlusions with other labels cannot occur, as visible areas are disjoint by nature. However, labels of adjacent visible areas can be placed closely, if the common edge is selected.

2.3 Computing Label Alignments

In the previous steps, we have computed the four visible edges, and selected one edge for each labeled object. In this section, we explain how the associated label is aligned to the selected edge, i.e., to the line sequence consisting of a tuple of 2D points or pixels. Single characters of a text label are translated separately by an offset vector. For the whole text, a list of offset vectors is computed. The start point of the text label is set to the start point of the edge, i.e., the furthest point on the left.

To generate the offset vector for the horizontally oriented edges (bottom and top edge), we collect all *relevant* edge points for each character (Fig. 4). An edge point is *relevant* for i th character, if it is located inside the character pixel bounding box $AABB_i$. From the relevant points, the average offset in y-direction is computed. For the vertically oriented edges (left and right edge), offsets are accordingly generated in x-direction.

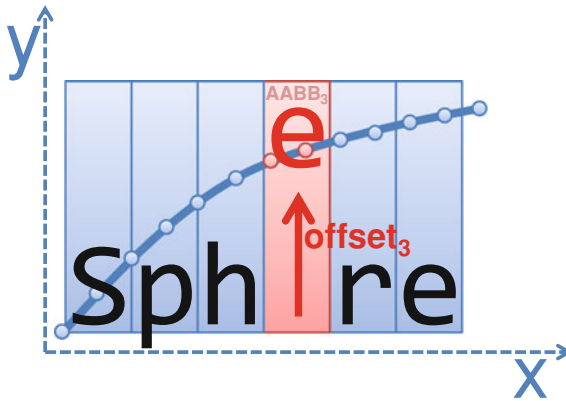


Fig. 4 Example offset computation: to align the character “e” of the label “Sphere” (3rd character) to the edge points (blue marked), the translation in y-direction, the *offset*, is computed. The *offset* is the average of the y-values of the edge points (red marked) inside the character bounding box $AABB$. The character position (x, y) of “e” (in image space) is then translated by the offset vector $(0, \textit{offset})^T$

Edge points do not necessarily need to be equidistant, if the sampling distance is higher than 1 pixel. Hence, a character can be located between two adjacent edge points without intersections. As a result, the translation offset between two adjacent characters can be very high, i.e., the character spacing between adjacent characters can be too loose. To enable sufficient character spacing for character translation in this case, we add the next adjacent edge points around the character bounding box. Additionally, we interpolate offsets using the arithmetic mean. As a result, we obtain a label alignment to a smoothed curve approximation of the edge.

The precision of the offset vector depends on the sampling distance. If the sampling distance is chosen too large, label alignments do not match anymore to the visible silhouette. Applying smaller sampling distances can lead insufficient performance. In our application scenarios with Full-HD resolution, we tested with a sampling distance of at least 10 pixels and maximally 50 pixels to provide reasonable label alignments with interactive frame rates.¹

If the whole text was transformed, legibility could be reduced due to text distortion. Hence, to avoid artifacts in text rendering, we use a font rendering technique that enables character transformation, i.e., single characters can be transformed independently. However, we also allow transformations of the whole text by translation or rotation. To enable real-time text alignment to curves, we implemented a font rendering technique that completely runs on the GPU using a single pixel shader with the following input textures: two font textures as well as a text texture and an offset texture per label. As these textures are low-resolution textures, the total GPU memory consumption is also low. The pixel shader for character transforming and rendering is implemented using the OpenGL Shading Language (GLSL) (Rost 2006).

The offset vectors are passed to the pixel shader as an additional 1D texture (*offset texture*). In the pixel shader, each character is translated by the according offset contained within the offset texture.

3 Conclusions

We presented a visibility-driven, silhouette-based labeling technique for 3D virtual worlds such as 2D and 3D maps that reduces occlusions of labeled objects by labels while fast assignment by placing labels close to labeled objects is preserved. Our technique enables dynamic text transformations around the visible silhouette of labeled objects using GPU-based font rendering that preserves text legibility. Our method provides detailed user-defined configuration, including (1) the *edges priorities*, i.e., the bottom edge is always preferred against the other edges, (2) the *label distance to the visible silhouette*, i.e., labels can be aligned on the silhouette

¹ We tested on the following hardware: Intel Xeon CPU 2 × 2.6 GHz, NVIDIA GeForce GTX 480.

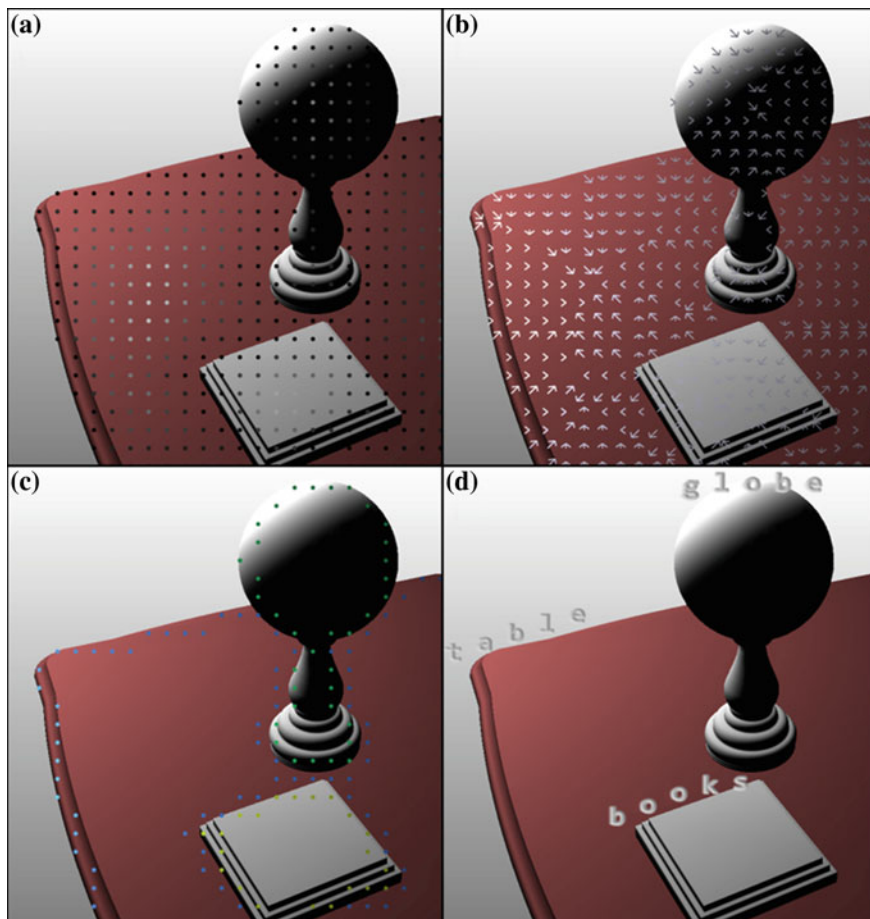


Fig. 5 Example for silhouette-based labeling applied to an indoor scene. From **a** to **d**: distance field, inner vector field, visible silhouettes, and aligned labels. In this example, we aligned labels to the *outer* visible silhouette of labeled objects and applied further smoothing to character translation

or to the inner/outer silhouette (Fig. 5), and (3) *label resizing*, i.e., labels are allowed/not allowed to be resized, if no edge is sufficiently sized.

Our labeling method projects labeled 3D objects into image space, i.e., 3D objects are represented as 2D areas in image space. Consequently, labeling of volumetric objects is reduced to labeling of areal objects (area feature labeling) to provide a generic approach for both 2D and 3D labeling. Kern and Brewer (2008) stated that area feature labeling is most difficult, so labels for areas must be placed first, followed by point and line feature labels. This ordering is mainly related to occlusions caused by point or line feature labels contained in area features, which are also labeled. With our approach, labels of objects, whose visible areas

degenerate to a point or a few points, are not depicted. They can be explored interactively by zooming in.

In our examples, we used a monospaced font type, i.e., all character glyphs have the same width. For a monospaced font type, the determination of offsets for edge alignment can be handled significantly more efficiently, in computational time and storage amount, than for a variable-width font type: for a variable-width font type, an additional offset texture is required that contains the character width information of the font type. Further, the pixel shader has to perform kerning and, thus, more texture look-ups. On the CPU, more complex distance computations for the offset vector have to be processed.

For future work, we want to compute the 2D skeleton of visible regions or *minor skeleton* (Freeman and Ahn 1984) respectively. Using the minor skeleton, we can align internal labels more precisely in the according visible regions. A real-time capable algorithm for image-based computation of the minor skeleton is provided by Telea and van Wijk (2002).

References

- Abdi H, Williams L (2010) Principal component analysis. *Comput Stat* 2(4):433–459
- Beauchemin SS, Barron JL (1995) The computation of optical flow. *ACM Comput Surv* 27(3):433–466
- Bekos MA, Kaufmann M, Nöllenburg M, Symvonis A (2010) Boundary labeling with octilinear leaders. *Algorithmica* 57(3):436–461 (Scandinavian Workshop on Algorithm Theory)
- Christensen J, Marks J, Shieber S (1994) An empirical study of algorithms for point feature label placement. *ACM Trans Graph* 14(3):203–232
- Freeman H, Ahn J (1984) AUTONAP—an expert system for automatic map name placement. In: *International symposium on spatial data handling*, pp 544–569
- Imhof E (1985) Positioning names on maps. *Cartogr Geogr Inf Sci* 2:128–144
- Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. *ACM Comput Surv* 31(3):264–323
- Kern J, Brewer C (2008) Automation and the map label placement problem: a comparison of two GIS implementations of label placement. *Cartogr Perspect* 60:22–45
- Kopetz H (1993) Should responsive systems be event-triggered or time-triggered? *Inst Electron Inf Commun Eng E76-D:1325–1332*
- Lehmann C, Döllner J (2012) Automated image-based label placement in interactive 2D/2.5D/3D maps. In: *Symposium on service-oriented mapping*
- Maaß S, Döllner J (2006) Dynamic annotation of interactive environments using object-integrated billboards. In: *14th international conference on computer graphics, visualization and computer vision*, pp 327–334
- Maaß S, Döllner J (2006) Efficient view management for dynamic annotation placement in virtual landscapes. In *6th international symposium on smart graphics 2006*, vol 4073. Springer, Heidelberg, pp 1–12
- Maaß S, Döllner J (2007) Embedded labels for line features in interactive 3D virtual environments. In: *AFRIGRAPH '07: proceedings of the 5th international conference on computer graphics, virtual reality, visualisation and interaction in Africa*, pp 53–59
- Mote, K. (2007). Fast point-feature label placement for dynamic visualizations. *Inf Vis* 6:249–260 (Data Structures and Algorithms)

- Raskin J (2000) *The humane interface: new directions for designing interactive systems*. ACM Press, New York
- Rost RJ (2006) *OpenGL(R) shading language*, 2nd edn. Addison-Wesley Professional, Boston
- Stein T, Décoret X (2008) Dynamic label placement for improved interactive exploration. In: NPAR (symposium on non-photorealistic animation and rendering), pp 15–21
- Telea A, van Wijk JJ (2002) An augmented fast marching method for computing skeletons and centerlines. In: *Symposium on data visualisation*
- Vaaranemi M, Treib M, Westermann R (2012) Temporally coherent real-time labeling of dynamic scenes. In: *3rd international conference on computing for geospatial research and applications*. ACM, pp 17:1–17:10
- Wagner F, Wolff A, Kapoor V, Strijk T (2001) Three rules suffice for good label placement. *Algorithm Special Issue GIS 2000*:334–349