Level-of-Detail Visualization of Clustered Graph Layouts

Michael Balzer Oliver Deussen

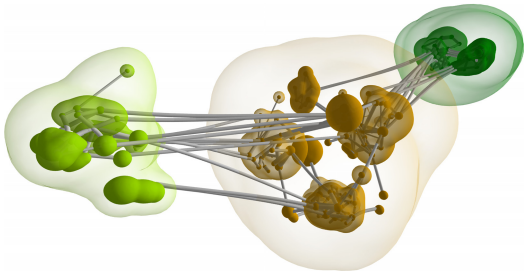
Department of Computer and Information Science
University of Konstanz, Germany

ABSTRACT

The level-of-detail techniques presented in this paper enable a comprehensible interactive visualization of large and complex clustered graph layouts either in 2D or 3D. Implicit surfaces are used for the visually simplified representation of vertex clusters, and so-called edge bundles are formed for the simplification of edges. Additionally, dedicated transition techniques are provided for continuously adaptive and adjustable views of graphs that range from very abstract to very detailed representations.

Keywords: Graph visualization, level-of-detail, clustered graphs, implicit surfaces, edge bundles.

Index Terms: I.3.6 [Methodology and Techniques]: Interaction Techniques; H.5.2 [Information Interfaces and Presentation]: User Interfaces;



1 INTRODUCTION

The visualization of graphs plays a vital role in the field of information visualization, and an enormous number of layout algorithms as well as a broad variety of visualization techniques exist. However, a major problem of these commonly known techniques is the limitation of current display devices, and the human cognitive system in general. A graph visualization containing more than a few hundred vertices and edges, results in incomprehensible representations with many overlappings and occlusions, which make a differentiation and a closer investigation of individual vertices and edges almost impossible. Furthermore, the performance limitations of graphics hardware restrict the interactivity of graph visualizations. In fact, real world data sets have grown by magnitudes and often consist of thousands or even millions of vertices and edges, such as in data of transportation networks, financial transactions, social networks of internet communities or software structures.

For the investigation of these large and complex data sets, it is necessary to provide abstracted views that reduce the amount of presented information and simultaneously limit the geometric complexity of the representation in order to enable interactivity. An obvious solution is an automatic or user-driven collapsing of parts of the graph. The disadvantage of this method is that the structure of the graph is modified, which often results in an unavoidable recomputation of the layout, and eliminates relevant information about the collapsed vertices and edges.

This paper presents a level-of-detail approach that only modifies the visual representation of the graph and does not alter the structure or the computed layout of the graph. Level-of-detail techniques are commonly used in computer graphics for the viewpoint-dependent rendering of objects, in which distant objects are rendered with less detail to decrease the polygon count. The approach presented in this paper uses level-of-detail to reduce the polygon count for interactively visualizing large and complex graphs, as well as to reduce the visual complexity of the information in general.

Level-of-detail techniques are based on the substitution of a complex object by a simplified object or the substitution of a group of objects by a single one. In the here presented level-of-detail application, groups of vertices and groups of edges are substituted. Consequently, the prerequisite is grouping information at different abstraction levels based on the spatial distribution of the vertices. Thus, this approach is specific to clustered layouts.

In the following Section 2, related work is addressed, and basic knowledge of clustered graphs and level-of-detail is imparted. Section 3 presents the level-of-detail approach for clustered graph layouts based on implicit surfaces and edge bundles. Finally, a discussion of the achieved results is given in Section 4.

2 BACKGROUND

2.1 Related Work

An enormous amount of work has been published in the field of graph drawing to generate meaningful visualizations of graphs—overviews can be found in [4, 14]. Most of previous as well as ongoing work concentrates on graph layouts, thus focusing on the spatial distribution of the vertices and the routing of the edges of the graph. The visualization itself is generated by drawing an object for each vertex at its computed positions and representing edges by direct line connections or curves. Visualizing large and complex graphs with these basic techniques result in ambiguous representations with many overlappings and occlusions.

Approaches that take advantage of meta data, like clustering or hierarchical information, are rare. If meta information is considered, mostly the structure of the graph is modified by collapsing groups of vertices to one single vertex and the connected edges respectively. Thereby relevant information about the collapsed subgraph is largely lost. Approaches that consider meta information and do not alter the structure and/or the layout of the graph are presented in [10, 6, 19, 7]. Here, hyperspheres, ellipsoids, or rectangles are wrapped around clusters of vertices, with the disadvantage that the vertex distribution within the clusters is not very well represented because of the non-adaptive shapes. A more advanced approach that uses implicit surfaces to extract and visualize vertex clusters in graphs is presented in [8, 18]. The thereby generated cluster representations are much more adaptive to the spatial structure of the contained vertices. Because of the focus on cluster extraction, this approach neither considers hierarchical clustering nor level-of-detail representations. Another approach [1] of the authors of the here presented paper already used implicit surfaces to emphasize two-dimensional graph structures, but without considering

multi-level clustering or level-of-detail representations.

Another approach for emphasizing the clustering of graph vertices is presented in [20], which uses spheres with constant size in screen space for the representation of vertices, whereby partly overlapping spheres visually abstract vertex clusterings. This method has the disadvantage that the visual sizes of the clusters strictly depend on the density of the clusters. Additionally, this visualization determines the clustering method itself, so that alternative clusterings can not be visualized.

An approach for the hierarchical abstraction of edges based on spline curves is presented in [11], which is related to the here presented edge bundle approach. The difference is that [11] uses point position for the edge deformation, whereas in the here presented method edge bundles are created based on the implicit surfaces of the clusters. Furthermore, this method does not avoid edge occlusions so that the size of the edge bundles can hardly be compared, and it does not support locally different levels of detail.

2.2 Clustered Graph Layouts

A *graph* $G = (V, E)$ consists of a finite set of vertices V and a finite set of edges E with $E \subseteq V \times V$. Each vertex $v \in V$ and each edge $e \in E$ may be attributed with a *weight* $w \geq 0$, if not, $w = 1$ is assumed. A *clustered graph* $C = (G, T)$ consists of a graph G and a rooted tree T such that the vertices of G are exactly the leaves of T. Each node n of T represents a *cluster* $V'(n)$ with $V'(n) \subseteq V$ that are the leaves of the subtree rooted at n. Thereby the rooted tree T describes an inclusion relation between clusters. The weight $w(n)$ of a cluster $V'(n)$ is the sum of the weights of the contained leaves of n. The *height* $h(n)$ of a node $n \in T$ is defined as the depth of the subtree of T rooted at n. A *n-dimensional layout* of a clustered graph $C = (V, E)$ is a vector of vertex positions (x_v) with $v \in V$ and $x_v \in \mathbb{R}^n$. Usually, graph layouts are embedded in $\mathbb{R}^2$ or $\mathbb{R}^3$. The structure of a clustered graph is illustrated in Figure 1.

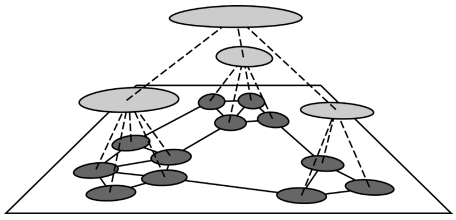


Figure 1: The Structure of a clustered graph.

2.3 Level-of-Detail for Graph Visualization

Level-of-detail in the context of computer graphics is used to represent a model or object with different resolutions depending on its distance to the viewer. The motivation is to decrease the polygon count, whereas the reduced visual quality of the model or object ideally remains unnoticed due to its smaller size as a result of its distance and the usage of a perspective projection.

In the context of graph visualization, polygon reduction is just a welcomed side effect that enables interactivity even for very large graphs. The important point is to reduce the amount of presented information, whereas in contrast to level-of-detail applications in computer graphics, visual changes are deliberate. The fundamental idea is to represent a cluster of vertices of a clustered graph by a single object depending on its distance to the viewer, while the appearance of this object is highly simplified to reduce the perceptual effort of the user. Each of these cluster representing objects allows an identification and recognition of the characteristics of the cluster.

Since neither the structure nor the layout of the underlying graph should be modified, the clustered graph is generated based on the spatial clustering of the vertices given by the layout. For the determination of the clusters it is possible to use conventional cluster analysis [5, 13], a visualization oriented approach [9, 18] or to utilize the hierarchical information that may be part of the data set. The choice of the clustering method is in principle not important for the application of the here presented level-of-detail techniques, but it has of course influence on the resulting visualization. Thus, it may be chosen according to the demand of the user and the application. For a description of the clustering strategy see Section 3.

Clustered graph layouts focus on the spatial grouping of vertices, whereas the edge routing or the minimization of edge crossing is of less importance. Hence, the representation of the edges is strictly constrained by the vertex positions in the layout. This means that an edge is simply visualized by a direct connection between two vertices and all edges between two clusters of vertices are represented as one aggregated edge that is also visualized by a direct connection between the clusters.

The handling of the changeover between different detail levels is of crucial importance for the rendering of level-of-detail representations. The naive way is to switch between two representations at a given distance threshold. This is adequate to conventional level-of-detail approaches in computer graphics, where the visual appearance of the model or object is similar for both detail levels. However, for an application with different appearances at different levels as presented here, this abrupt change would distract the user. A continuous change between the detail levels is necessary to preserve the mental map of the user and to enable the comprehension of the abstraction process in the graph visualization.

3 Contribution

This section describes the approach for the level-of-detail visualization of clustered graph layouts. A discussion of the hierarchical clustering strategy are given in the next subsection. Subsections 3.2 and 3.3 separately describe the level-of-detail representations for the clusters and the edges of the clustered graph, including the different handling of the changeover between the detail levels. Again it must be emphasized, that this level-of-detail approach for clustered graph layouts is not restricted to special clustering methods and can be used for two- and three-dimensional layouts.

3.1 Hierarchical Clustering Strategy

To generate the clustered graph C from the graph G, which has a given layout, a root node n_r is defined for the rooted tree T that contains the complete set of vertices V of the graph G. Then V is partitioned into the subsets $\{V_1, ..., V_j\}$ that form the clusters $\{V'(n_1), ..., V'(n_j)\}$ of $\{n_1, ..., n_j\}$ as children of n_r in T. Afterwards, each of these clusters $V'(n_i)$ with $1 \leq i \leq j$ is again partitioned recursively into smaller clusters as their according child clusters in T. This recursion stops if no further partitioning of a subset of vertices is desired.

The selection of the clustering algorithm and its parameters may vary with regard to the application. For example, the resulting clusters could roughly contain the same number of vertices or the clusters could enclose almost equal areas in 2D or volumes in 3D within each hierarchy level etc. Likewise the resulting rooted tree T does not have to be balanced, rather it can have any structure that is useful for the given layout and the targeted application.

3.2 Cluster Representation Using Implicit Surfaces

For the visualization of the vertices of the graph, it is assumed that they already possess a given visual representation, which is not changed in the level-of-detail enhancement. Additionally, objects are added that represent the clusters of the graph. By visualizing a cluster by one single object, it is important to represent the distribution and attributes of the contained vertices as precisely as possible. If the distribution of vertices in the cluster is shaped like a sphere,

the representation of the cluster should look like a sphere; if it is shaped like a cylinder or like a torus, the representation of the cluster should also look like a cylinder or a torus. Thus, for perceptual reasons it is not adequate to use just one single primitive or even a set of primitives for the representation of the clusters. Rather the representations of the clusters must be directly generated out of the contained sub-clusters or vertices in order to allow statements about the properties of the cluster, even if it is shown from a distant viewpoint. A very adaptive concept for complex shapes is offered by implicit surfaces based on density fields defined by generator sets. This concept is explained in the subsequent Section 3.2.1, starting with the simple case of points as generators and then extending to arbitrary generator objects. Section 3.2.2 illustrates the application of implicit surfaces for graph cluster visualization, and Section 3.2.3 describes their level-of-detail representation.

3.2.1 Implicit Surfaces

Following the Metaball concept presented in [17], an *implicit surface s* can be described by a set of generator points P, whereby each generator point $p_i \in P$ has a radius of influence r_i. The influence of a single generator point p_i at a point q is described by a density function $\mathscr{D}_i(q)$ defined as

$$\mathscr{D}_i(q) = \begin{cases} \left(1 - \left(\frac{\|q-p_i\|}{r_i}\right)^2\right)^2, & \text{if } \|q-p_i\| < r_i \\ 0, & \text{if } \|q-p_i\| \geq r_i \end{cases} . \quad (1)$$

The summation of the density function for all generator points forms the density field $\mathscr{F}$ as

$$\mathscr{F}(q) = \sum_i \mathscr{D}_i(q) - \tau \quad (2)$$

with $\tau \geq 0$. Thus, the implicit surface s with $\mathscr{F}(q) = 0$ is defined as those points q where the sum of the density values of all generators equals the threshold τ. Note that any vector norm can be used for the distance computation in Equation 1. For all examples in this paper, the Euclidian norm is used. Figure 2 shows an implicit surface defined in $\mathbb{R}^2$ by two generator points and a threshold of $\tau = 0.3$.

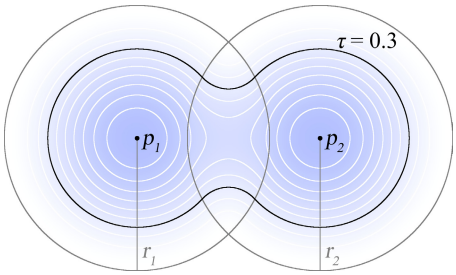


Figure 2: An implicit surface defined in $\mathbb{R}^2$ by two generator points p_1 and p_2 with radii of influence r_1 and r_2 and a threshold of $\tau = 0.3$.

The implicit surface concept based on generator points can be easily extended to arbitrary generator objects by substituting the distance computation between the point q and the generator p_i in Equation 1. The radius of influence r_i for each generator p_i is preserved and the computation of the density field is the same as in Equation 2. This generalization allows to compute implicit surfaces for any generator shapes, such as spheres, lines or even polygonal objects—an example is presented in Figure 3. Depending on the application, volume enclosing generators, e.g. spheres, may be considered as solid objects, where the interior has a distance of 0 or as hulls with standard distance computation to the surface of the generator.

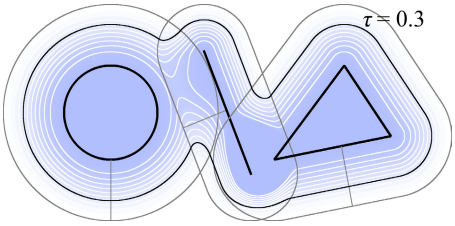


Figure 3: An implicit surface defined in $\mathbb{R}^2$ by a circle, a line, and a triangle with a threshold of $\tau = 0.3$.

The general shape of an implicit surface can be influenced by the global threshold τ in Equation 2. Greater values describe a closer modulation of the implicit surface to its generator set. Values of $\tau > 1$ result in surfaces that do not necessarily enclose the generator set, and should therefore be avoided. Furthermore, a global user defined parameter that is multiplied with the radius of influence of every generator can be introduced. This also allows to control the modulation of the surface to its generators, by which greater values result in surfaces that enclose larger volumes and modulate less to the generators.

The extraction of implicit surfaces is performed by the Marching Cube algorithm [15]. Its result is a polyline in 2D or a triangle mesh in 3D, which approximates the implicit surface at a regular grid with a user specified resolution. Especially for 3D, an additional mesh optimization [12] may be applied to the resulting surface approximation to obtain a homogeneous mesh with a lower number of primitives and better vertex normals. By choosing appropriate parameters for this re-meshing, the changes of the surface topology are not noticeable for the user.

3.2.2 Graph Cluster Surfaces

The utilization of implicit surfaces allows us to generate clustered graph visualizations by a bottom-up approach:

First, an implicit surface s is generated for each cluster $V'(n)$ of a node n in the rooted tree T that has an height of $h(n) = 1$, which means that the children of n are vertices. Thereby the graphical representations of these vertices form the generator set P of the implicit surface s. Any geometric input is possible as generator, ranging from simple points, circles or spheres, to complex polygonal shapes. The implicit surface perfectly adapts to its generating vertex representations. The position of the generators is given by the graph layout. The radius of influence r_i of each generator is derived from the weight w_i of the associated vertex. According to experience, the following formula provides good results, where $\delta = \pi$ is used for all examples in this paper.

$$r_i = \sqrt{\frac{w_i}{\delta}} \cdot \delta \quad (3)$$

To generate the implicit surfaces of clusters at higher levels, the graphical representations of the vertices could also be used as generators. Since the radius of influence of each generating vertex had to be increased level by level, the result would be more and more sphere like clusters that do not preserve the distribution of the vertices. Instead, the implicit surfaces of the child clusters $\{V'(n_1), ..., V'(n_j)\}$ of the node n according to the rooted tree T are used as the generators for the implicit surface representation of the cluster $V'(n)$. Thus, implicit surfaces are generated out of other implicit surfaces. This allows to identify the shapes and the distribution of the sub-clusters, and at the same time preserves a homogeneous representation at every detail level. The position of the generator clusters is given by the underlying vertex positions of

the layout. The radius of influence is computed by means of Equation 3, where the weight of a cluster w_i is the sum of the weights of its contained vertices. Note that the implicit surface s of a cluster may not be cohesive, but may rather consist of a set of sub-surfaces $\{s'_1,...,s'_n\}$ with $s'_i \subseteq s, 1 \leq i \leq n$, depending on the degree of disruption of the cluster. Several examples for clustered graph representations by implicit surfaces are given at the following pages.

The specific algorithms for the distance computation in Equation 1 result from the individual shape of the generator objects, in which the generators are treated as solid objects with an interior distance of 0. The visual representation of the vertices is usually very simple, so that their interior test and distance computation should be fast. In contrast, the implicit surface representations of the clusters normally consists of a few hundred line segments in 2D or a few thousand triangles in 3D, for which the interior test and the distance computation is quiet expensive. By using hierarchical methods, like binary space partitioning or bounding volume hierarchies, for the distance computation of implicit surfaces, this performance issue is largely eliminated.

3.2.3 Level-of-Detail Representation

The smooth transition between the different detail levels is of great importance for a comprehensive visualization and for preserving the mental map of the user while navigating within the clustered graph. For the changeover, the transparency approach presented in [2] is applied and generalized, where the opacity of an object adapts dynamically to the position of the viewpoint. Therefore the clustered graph is rendered from top to bottom. First, the root node $n_r \in T$ is rendered, and if n_r has an opacity of $o_r < 1$ then the child nodes $\{n_1,...,n_j\}$ of n_r are rendered as well. If each child node n_i with $1 \leq i \leq j$ again has an opacity of $o_i < 1$ then the child nodes of n_i are rendered accordingly, and so on.

For the determination of the opacity the following procedure is used: Let S be the set of implicit surfaces that represent the clusters in a clustered graph. The point c_i is the center of mass and d_i the diameter of the surface $s_i \in S$, where d_i is specified by the average of the differences between the minimum and maximum coordinate values of s_i in each dimension. Additionally, two global parameters σ and ρ are introduced, which are chosen by the user. The opacity o_i of s_i with regard to the current viewpoint v is then computed by

$$o_i = \begin{cases} 0, & \text{if } \overline{o}_i \leq 0 \\ \overline{o}_i, & \text{if } 0 < \overline{o}_i < 1 \\ 1, & \text{if } 1 \leq \overline{o}_i \end{cases} \quad \text{with} \quad \overline{o}_i = \frac{\|c_i - v\| - d_i \cdot \sigma}{d_i \cdot \rho}. \qquad (4)$$

The global parameter σ influences the position of the distance interval in which the change of the opacity takes place, while the global parameter ρ has an influence on the size of the interval. If $o_i = 1$, then the implicit surface s_i is drawn fully opaque. A semi-transparent drawing with a transparency $t = 1 - o_i$ is applied if $0 < o_i < 1$. For opacity values of $o_i = 0$ the drawing of the implicit surface s_i is omitted. Figure 4 additionally illustrates this procedure and the influence of the global parameters σ and ρ on the opacity o_i. An example with different levels of detail based on this opacity computation is given in Figure 8 at the end of the paper.

3.3 Edge Representation Using Edge Bundles

The clusters of a clustered graph C are obtained from by the set of vertices V of the underlying graph G. The set of edges E of G usually does not affect the cluster computation. Therefore, it exists no counterpart of edge clusters in the layout of a clustered graph. In fact, the vertices that are connected by an edge may be part of the same sub-cluster or may be far away in the layout as well as in the cluster hierarchy. Nevertheless, it is possible to generate aggregated edge representations by utilizing the cluster information of the vertices. Therefore, an edge is not just considered as a direct connection between two vertices, rather it is routed according to the

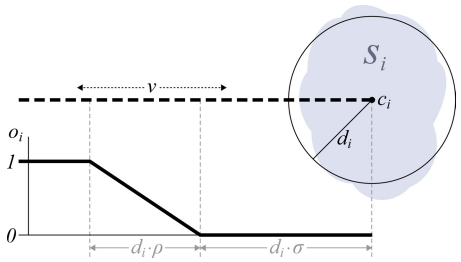


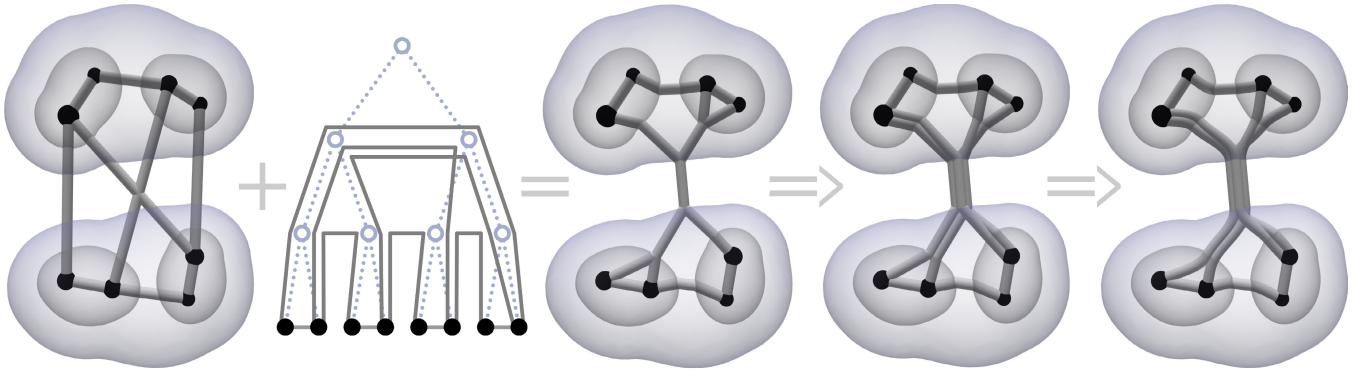
Figure 4: The opacity $o_i \in [0,1]$ of an implicit surface s_i depends on the center c_i and the diameter d_i of s_i, the distance between c_i and current viewpoint v, and the global parameters σ and ρ.

clusters of the graph. This routing information is then used to form edge bundles, which group edges according to the cluster hierarchy. Based thereon, the visible parts of the edges are extracted with regard to the current viewpoint, thereby providing smooth transitions between the different detail levels. The following Section 3.3.1 describes the edge routing, and the preceding Section 3.3.2 presents the approach of edge bundle. Finally, their level-of-detail representation is explained in Section 3.3.3.

3.3.1 Edge Routing

To generate the routing of an edge $e \in E$ of the clustered graph C that connects the vertices $v_1, v_2 \in V$, first the parent node $n_p \in T$ of C is determined, which has both v_1 and v_2 as its leaves, whereby no other node $n \in T$ with a height of $h(n) < h(n_p)$ exists that has v_1 and v_2 as its leaves. We also say that the node n_p is the owner of the edge e. Afterwards, we create an ordered set of nodes of $N \subseteq T$ in two steps. In the first step, by starting at v_1 and ascending in the rooted tree T, each visited node n_i with $h(n_i) < h(n_p)$ is inserted in the ordered set according to the ascent sequence. In the second step, by starting at v_2 and again ascending in the rooted tree T, each visited node n_i with $h(n_i) < h(n_p)$ is inserted in the ordered set, but now according to the reversed ascent sequence. As result, the edge $e = (n_p, N)$ is now defined by a node $n_p \in T$, which is the owner of e, and an ordered set $N = (v_1, n_1, ..., n_j, v_2)$ with $N \subseteq T$ and $n_p \notin N$. This edge routing is further outlined in Figure 5(b).

The ordered set of nodes N in the definition of an edge $e = (n_p, N)$ describes the routing of e between the vertices v_1, v_2, which are the first and the last element of N, as edge segments between neighboring nodes in N. The notation $n_i \prec n_j$ with $n_i, n_j \in N$ denotes that n_i is a predecessor of n_j in the ordered set N. For each node $n \in N$ a so-called *port* is computed, which specifies an endpoint of the edge segments of e in the graph visualization. Note that each node may not only have one port, but rather many ports. The computation of the ports is based on the clustering and the location of v_1 and v_2 as a top-down approach according to the rooted tree T. First, the nodes $n_1, n_2 \in N$ with $v_1 \preceq n_1 \prec n_2 \preceq v_2$ in N, and $h(n_1) > h(n_i), h(n_2) > h(n_i), h(n_1) = h(n_2)$ are identified, where $n_i \in N$ and $n_i \neq n_1, n_i \neq n_2$. The nodes n_1 and n_2 are necessarily children of n_p. The edge segment between n_1 and n_2 is located at a higher level of T than all other segments of e. The ports of n_1 and n_2 are defined by the endpoints of the shortest connection between the representations of n_1 and n_2 that contain v_1 and v_2 respectively. This means that if a node n is a leaf of T, and thereby a vertex, the representation of n containing the vertex v is the visual representation of v. If a node n is an inner node of T, and thereby a cluster of vertices that is visualized by an implicit surface s, then the representation of n containing the vertex v is the sub-surface $s' \subseteq s$ that encloses v. Starting from n_1 and n_2, the ports of the other nodes of N are now determined by descending T separately towards v_1 and v_2 according to the ordered set N. Therefore, n_1 and its predeces-



(a) straight edges (b) edge routing tree (c) routed edges (d) edge bundle (e) smoothed edge bundle

Figure 5: Edge Bundle Generation: The existing (a) straight edges are routed according to the (b) cluster tree to obtain (c) routed edges that incorporate the hierarchy of the clusters. This routing information enables the generation of (d) edge bundles, which may additionally be (e) smoothed by using Bèzier curves. This allows for an easier tracking of the individual edges, and enables smooth level-of-detail transitions.

sor $n_{\prec}$ in N are taken, then the port of $n_{\prec}$ is defined by the new endpoint of the shortest connection between the port of n_1 and the representation of $n_{\prec}$ containing v_1, and after all, n_1 is substituted by $n_{\prec}$. This is repeated until n_1 has no predecessor in N. The same procedure is performed with n_2 and its successor $n_{\succ}$ accordingly towards v_2.

The result is an ordered set of ports for each edge that enables the drawing of edge segments that are routed correspondent to the implicit surfaces of the clusters. Figure 5(c) further illustrates the edge routing and clarifies that the route of an edge is its shortest path between two nodes in the cluster tree. This routing information is now used to form edge bundles as described in the next section.

3.3.2 Edge Bundling

After performing the edge routing for the complete clustered graph, many edge segments use the same ports. Thereby edge segments between the same two ports are occluding each other. Thus the information about the number, the weight, and the structure of the edges is lost for these edge segments. The idea for eliminating this occlusion is to form so-called *edge bundles*, with the individual edges as their fibres. A real world analogy for an edge bundle is a cable loom that is tied up out of single cables (the edges) at different branching points (the ports). Within each edge bundle segment, the contained edges are close-packed with minimal intersection between neighbors. The aggregated weight of the edges between two ports is thereby reflected by the weight of the according segments of the edge bundle.

An *edge bundle* $b = (E, B')$ consists of a set of edges E and a set of edge bundle segments B'. An *edge bundle segment* $b'_{p_1 \leftrightarrow p_2} = (p_1, p_2, E')$ is defined by two ports p_1, p_2, and a set of edge segments E', whereby $b'_{p_1 \leftrightarrow p_2} = b'_{p_2 \leftrightarrow p_1}$. Each edge bundle segment $b'_{p_i \leftrightarrow p_j}$ has a weight $w_{p_i \leftrightarrow p_j}$, which is the sum of the weights of the contained edge segments. The weight of an edge segment is equal to the weight of the edge the segment belongs to.

An edge bundle $b_p = (E, B')$ is generated for each pair of child nodes $\{n_1, n_2\}$ of the node $n_p \in T$ that are connected by a set of edges $E = \{e_1, ..., e_n\}$ with $e_i = (n_p, N_i), 1 \leq i \leq n$, and $n \geq 2$. The node n_p is thereby necessarily the owner of the edges in E and also the owner of the edge bundle b_p. This means that the edge bundle b_p is only drawn if the implicit surface s_p of the node n_p has an opacity of $o_p < 1$, so that the interior of n_p is drawn. This mechanism highly reduces the amount of drawn edges in the visualization. Furthermore, the set B' of b_p contains all edge bundle segments $b'_{p_i \leftrightarrow p_j} = (p_i, p_j, E')$ that connect the ports p_i and p_j by an edge segment of an edge $e \in E$. The set E' of $b'_{p_i \leftrightarrow p_j}$ contains all edge segments that connect p_i and p_j.

The individual endpoint positions of the edge segments are computed bottom-up as a translation in relation to the position of the corresponding port, which is additionally outlined in Figure 6. In the first step, the translation vectors are determined for each port p_0 of a leaf node n_0, which means that p_0 is located at a vertex and $h(n_0) = 0$. Therefore, a translation vector is computed considering all edge segments that are contained in the set $E'_{p_0 \leftrightarrow p_1}$ of the edge bundle segment $b'_{p_0 \leftrightarrow p_1} = (p_0, p_1, E'_{p_0 \leftrightarrow p_1})$. In the second step, the translation vectors are computed for each port p_1 of the node n_1 with $h(n_1) = 1$, which means that p_1 is located at a cluster. In contrast, the translation vectors are now determined as a summation of the already computed translation vectors of the first step at the port p_i, and of a translation vector for each edge bundle segment $b'_{p_i \leftrightarrow p_1} = (p_i, p_1, E'_{p_i \leftrightarrow p_1})$, where p_i is a port of the node n_i with $h(n_i) = 0$. In the third step, the translation vectors are computed for each port p_2 of the node n_2 with $h(n_2) = 2$ accordingly. Now the translation vector of each edge segment are determined similarly to the second step as a summation of the translation vectors at the port p_i, and of a translation vector for each edge bundle segment $b'_{p_i \leftrightarrow p_2} = (p_i, p_2, E'_{p_i \leftrightarrow p_2})$, where p_i is a port of the node n_i with $h(n_i) = 1$. This iterative computation continues while a further ascent in T is possible. In each step after the first one, the translations are a combination of the translation of the edge bundles of the lower level and the already computed translations of the lower level ports.

The computation of the individual translation vectors at the ports for 2D is a non-overlapping side by side placement of the edge segments at a linear cross section of an edge bundle. A detailed description of this trivial placement procedure is omitted. In 3D, this computation can be understood as a circle packing problem [3] for a circular cross section of an edge bundle, and is implemented as an adaption from [16] as follows: The set of circles C is derived from the considered set of edge segments E' or from the considered set of edge bundle segments B'. The radius r_i of a circle $c_i \in C$ is defined by the weight of the according edge segment in E' or edge bundle segment in B'. The center p_i of a circle $c_i \in C$ relates to the translation value of the single endpoint of the according edge segment in E' or relates to the endpoints of the edge segments of the according edge bundle segment in B'. The circle packing procedure determines the center p_i for each circle $c_i \in C$, while the radius r_i of c_i is fixed. Therefore, the set of circles C is sorted according to the radius r_i of each circle $c_i \in C$ in descending order. Afterwards, the circles in C are laid out in concentric bands. Therefore, the largest circle c_0 is placed in the center. Then a new band around c_0 is created with its width equal to the diameter of c_1, which is the

second-largest circle in C. This band is then filled with the next m circles $\{c_1,...,c_m\} \subseteq C$ in their given order side by side until the band is full. The remaining circles $\{c_{m+1},...,c_n\} \subseteq C$ are laid out similarly within the necessary number of new bands. In some cases with a small set of circles c, sometimes a layout within one single band with an inner radius of 0 is equally or more compact than a layout with a center circle. Lastly the layout is translated so that an enclosing circle located in the origin completely encloses the set of circles C while having a minimal radius. This circle packing scheme is further illustrated in Figure 6.

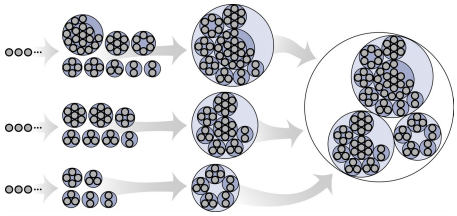


Figure 6: Circle Packing of Edge Bundles: The edges are laid out by a circle packing algorithm at each branching point of an edge bundle. This computation is performed bottom-up within the edge routing tree by using the layout of lower levels as input circles in higher levels.

The result is a distribution of the circles defined by their center, which is interpreted as translation vectors for the determination of the endpoints of the edge segments of an edge bundle segments. If an edge bundle is drawn with these values, the weight of the edge bundle segments does most likely not appear as the sum of the weights of the contained edge segments. This is a consequence of the gaps in the circle layout in 3D as well as of the often non-linear mapping between the weight and the visual size of the edges, either in 2D or 3D. The straightforward solution is to scale the translation vectors such that the size of each edge bundle segment has the identical size of the aggregated weights of the contained edge segments. This entails an overlapping of the single edge segments within each edge bundle segment, which is acceptable for aimed purpose.

By finally combining the different translation vectors to the final endpoint positions of the edge segments, the result are edge bundles that are routed according to the cluster tree as shown in Figure 5(d). Each edge bundle segment thereby provides a general idea of the structure and the weights of the contained edge segments. Additionally, a 'smoothing' may be applied, to eliminate the angular appearance of the edge bundles as shown in Figure 5(e). Therefore, each edge segment is not drawn as a direct connection between its endpoints p_1 and p_2, rather it is represented as a cubic Bézier curve, which is further described as part of the level-of-detail representation in the next section. These smooth edge bundles allow for an easier tracking of the individual edges within an edge bundle, and enable smooth transitions between the different views.

3.3.3 Level-of-Detail Representation

By now having the segmented edges with the individual port information, which contains the according endpoint and the associated cluster or vertex, it is possible to extract an edge visualization that considers the visibility of the clusters and vertices for all potential viewpoints. The intention is to treat every vertex or opaque cluster in the same way. This means that edges of the graph are considered as interconnections between visible nodes, which can be clusters or vertices. Non-visible parts of the edge are omitted. Because of the fact that the opacity of the clusters, and thereby also the visibility of other clusters and the vertices, is changing continuously while navigating through the visualization, the topology of the interconnections is changing continuously as well. Hence, similar to the vertex and cluster representation, a smooth transition between the different interconnection states at different detail levels is crucial.

Both, the determination of the interconnections for each viewpoint, and the transition between two different interconnection topologies, is based on the opacity of the implicit surfaces of the clustered graph. Therefore, an edge transition value t_i is determined for each visible node n_i of the rooted tree T. If n_i is a vertex then $t_i = 1$; if n_i is a cluster then t_i is calculated in dependency on the current opacity o_i of its implicit surface s_i by the following formula:

$$t_i = \begin{cases} 0, & \text{if } o_i \leq 0.5 \\ 2 \cdot o_i - 2, & \text{if } o_i > 0.5 \end{cases} \quad (5)$$

Thereby the linear gradient of the opacity in the interval $[0,1]$ is mapped to a linear gradient of the edge transition values in the interval $[0.5,1]$. The reason for this mapping is to have a faster level-of-detail changeover for the edges than for the clusters, which is according to experience more comprehensible for the user.

By having an actual edge transition value t_i for each visible node n_i, the visible edge bundles are visualized by separately drawing each of their edges. This procedure is described in the next paragraphs, where $e = (n_p, N)$ denotes an edge of the edge bundle $b = (E, B')$ with $e \in E$ that connects the vertices v_1, v_2.

Before the actual drawing step is executed, it is necessary to extract the relevant edge segments of e considering the current viewpoint. Therefore, in the first step, it is determined between which nodes $n_1, n_2 \in N$, with $v_1 \preceq n_1 \prec n_2 \preceq v_2$ in N, an interconnection exists. Thereby the node n_1 is the node that contains the vertex v_1, whereas all other other nodes $n_i \in N$ that contain v_1 have either an edge transition value of $t_i < 1$ or an height of $h(n_i) < h(n_1)$. The same applies to n_2, which contains v_2. These nodes n_1, n_2 are the only visible nodes in N with a transition value $t_i = 1$ considering the current viewpoint, and thus define the visible part of the edge. In the second step, it is determined between which nodes $n'_1, n'_2 \in N$, with $n_1 \preceq n'_1 \prec n'_2 \preceq n_2$ in N, a direct link exists. Thereby the node n'_1 contains v_1, whereas all other nodes $n_i \in N$ that contain v_1 have either an edge transition value of $t_i = 0$ or an height of $h(n_i) > h(n'_1)$. The same applies to n'_2, which contains v_2. This means that the implicit surface of any node $n_i \in N$, with $n'_1 \prec n_i \prec n'_2$, is not visible.

After the relevant edge segments have been determined, the final positions of each endpoint of the considered edge segments with regard to the current viewpoint can be determined—see also Figure 7. The final positions f_1 and f_2 for the endpoints at the nodes n_1 and n_2 are the positions of their ports p_1 and p_2. Each other final position f_i, where $f_i \neq f_1, f_i \neq f_2$, is a linear interpolating between the position of the according port p_i of the node n_i, and the intersection point p'_i of the direct link from n_1 to n_2 with the implicit surface s_i of the node n_i, based on the opacity o_i of s_i by $f_i = o_i \cdot p_i + (1 - o_i) \cdot p'_i$. The exact route of the interconnection of the edge e is specified by these final positions of the edge segments of e, so that now the interconnection can be drawn. The part of the edge e between n'_1, n'_2 is drawn as a direct link, e.g. a straight line or cylinder segment. The parts between n_1, n'_1 and n'_2, n_2 are drawn as Bèzier curves according to the edge segments of the edge e between n_1, n'_1 and n'_2, n_2 respectively. An Bèzier curve $B(t) = \{P_0, P_1, P_2, P_3\}$ is thereby specified by its control points P_0, P_1, P_2, P_3. These control points are defined by the endpoints f_1, f_2 of the considered edge segment, their edge transition values t_1, t_2, and their preceding endpoint f'_1 and succeeding endpoint f'_2, if any, by $P_0 = f_1$, $P_3 = f_2$,

$$P_1 = \begin{cases} f_1 + \frac{t_1 \cdot (f_2 - f_1) + (1 - t_1) \cdot (f_1 - f'_1)}{3 \cdot \|f_1 - f_2\|}, & \text{if } f'_1 \text{ exists} \\ f_1 + \frac{f_2 - f_1}{3}, & \text{if } f'_1 \text{ does not exist} \end{cases}, \quad (6)$$

$$P_2 = \begin{cases} f_2 + \frac{t_2 \cdot (f_1 - f_2) + (1 - t_2) \cdot (f_2 - f'_2)}{3 \cdot \|f_1 - f_2\|}, & \text{if } f'_2 \text{ exists} \\ f_2 + \frac{f_1 - f_2}{3}, & \text{if } f'_2 \text{ does not exist} \end{cases}. \quad (7)$$

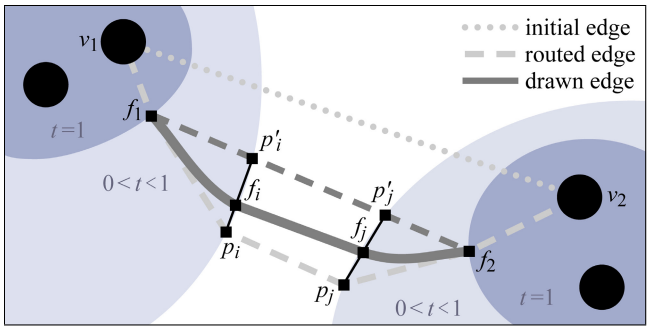


Figure 7: Edge Drawing: The visibility of the clusters determines the visible interconnection of the edge. The endpoint of an edge segments f_i is a linear interpolation between its port p_i, and the intersection point p'_i of the corresponding implicit surface with the direct link between the ports of the two opaque clusters with $t = 1$.

An example for the rendering of edges at different detail levels based on the opacity of the clusters is given in Figure 8.

4 Discussion

This paper presents an approach for the level-of-detail representation of clustered graph layouts. The usage of implicit surfaces for the visual simplification of graph clusters enables a comprehensible interactive visualization of the vertices and their distribution within the clusters. Depending on the viewpoint of the user, a graph cluster is thereby represented just as an abstract surface that allows basic statements about its properties or it reveals its contained subclusters and vertices respectively. This results in a visualization that does not show a cluttered mass of thousands of vertices, but rather presents a manageable number of abstracted clusters and in detail drawn vertices. Edge bundles are proposed for a comprehensible visualization of the edges between the vertices and clusters. Again, depending on the viewpoint, these bundles present aggregations of an arbitrary number of edges between the currently visible vertices and clusters, while simultaneously reflecting the structure and the weights of the represented edges.

Furthermore, smooth changeover techniques are provided for the visualization of vertices and clusters as well as for the edges of the graph, to enable continuous transitions between different views and abstraction levels of the visualization. These changeovers are both based on the individual opacity values of the implicit surfaces of the clusters, which depend on the distance from the corresponding cluster to the viewpoint. The opacity values are influenced by two global parameters that allow the user to easily manipulate the level of abstraction and the duration of the changeover independently.

The future work of the authors will address improvements for the arrangement of edges within the edge bundles. Currently, a straightforward circle packing solution with a short runtime is implemented. Its drawback are suboptimal positions of the individual edge segments that sometimes produce too large gaps or crossings between different edge bundle segments. Methods of resolution are an energy-based relaxation and the analysis of the connected nodes.

Finally, an example of a complex graph is presented in Figure 9. This real world graph contains more than 1500 vertices and 1800 edges within 126 clusters, and represents inheritance relations within a large software system. It can be explored in real-time with high frame rates, whereby the preprocessing step for the computation of the implicit surfaces and the edge bundles required less than one minute.

References

[1] M. Balzer and O. Deussen. Exploring relations within software systems using treemap enhanced hierarchical graphs. In *Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis 2005*, pages 89–94. IEEE Computer Society, 2005.

[2] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz. Software landscapes: Visualizing the structure of large software systems. In *Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 261–266. Eurographics Association, 2004.

[3] J. H. Conway and N. Sloane. *Sphere Packings, Lattices, and Groups*. Springer-Verlag, 1999.

[4] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[5] B. S. Duran and P. L. Odell. *Cluster Analysis: A Survey*, volume 100 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, 1974.

[6] T. Dwyer and P. Eckersley. Wilmascope - an interactive 3d graph visualisation system. *Lecture Notes in Computer Science*, 2265:442, 2002.

[7] Y. Frishman and A. Tal. Dynamic drawing of clustered graphs. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 191–198. IEEE Computer Society, 2004.

[8] M. Gross, T. C. Sprenger, and J. Finger. Visualizing information on a sphere. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 11–16. IEEE Computer Society, 1997.

[9] B. Heckel and B. Hamann. Visualization of cluster hierarchies. In R. F. Erbacher and A. Pang, editors, *Proceedings of SPIE Conference on Visual Data Exploration and Analysis*, volume 3298, pages 162–171, 1998.

[10] R. J. Hendley, N. S. Drew, A. M. Wood, and R. Beale. Case study: Narcissus: visualising information. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 90–96. IEEE Computer Society, 1995.

[11] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. In *Proceedings of the IEEE Symposium on Information Visualization*. IEEE Computer Society, 2006.

[12] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 19–26. ACM Press, 1993.

[13] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[14] M. Kaufmann and D. Wagner, editors. *Drawing Graphs: Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001.

[15] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.

[16] T. Munzner. H3: Laying out large directed graphs in 3d hyperbolic space. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 2–10. IEEE Computer Society, 1997.

[17] S. Murakami and H. Ichihara. On a 3d display method by metaball technique. *Transactions of the Institute of Electronics, Information and Communication Engineers*, J70-D(8):1607–1615, 1987. In Japanese.

[18] T. C. Sprenger, R. Brunella, and M. H. Gross. H-blob: a hierarchical visual clustering method using implicit surfaces. In *Proceedings of the Conference on Visualization*, pages 61–68. IEEE Computer Society Press, 2000.

[19] T. C. Sprenger, M. H. Gross, A. Eggenberger, and M. Kaufmann. A framework for physically-based information visualization. In *Proceedings of Eurographics Workshop on Visualization '97*, pages 77–86, 1997.

[20] F. van Ham and J. van Wijk. Interactive visualization of small world graphs. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 199–206. IEEE Computer Society, 2004.

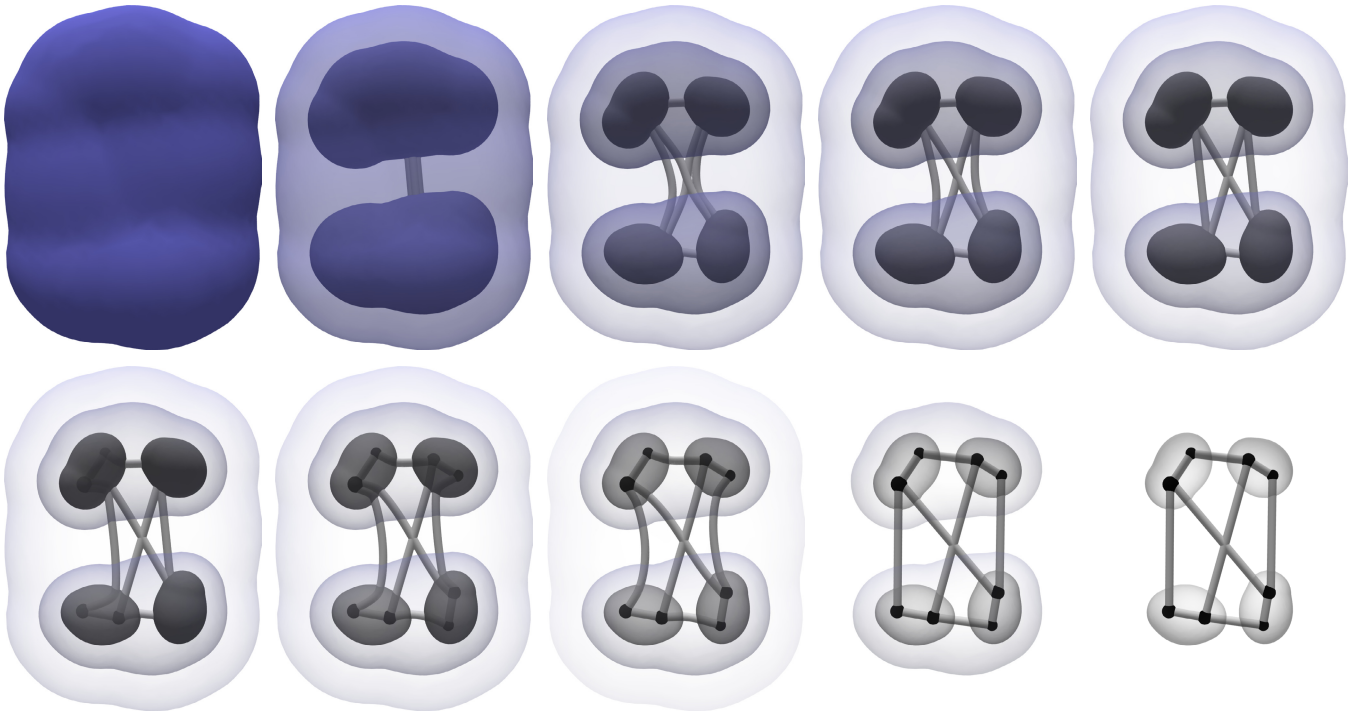


Figure 8: Different Levels of Detail: The opacity of the clusters determines the visibility of the contained clusters, vertices, and edges. Furthermore, it influences the topology and the exact route of the edges. This approach enables seamless transitions between the detail levels.

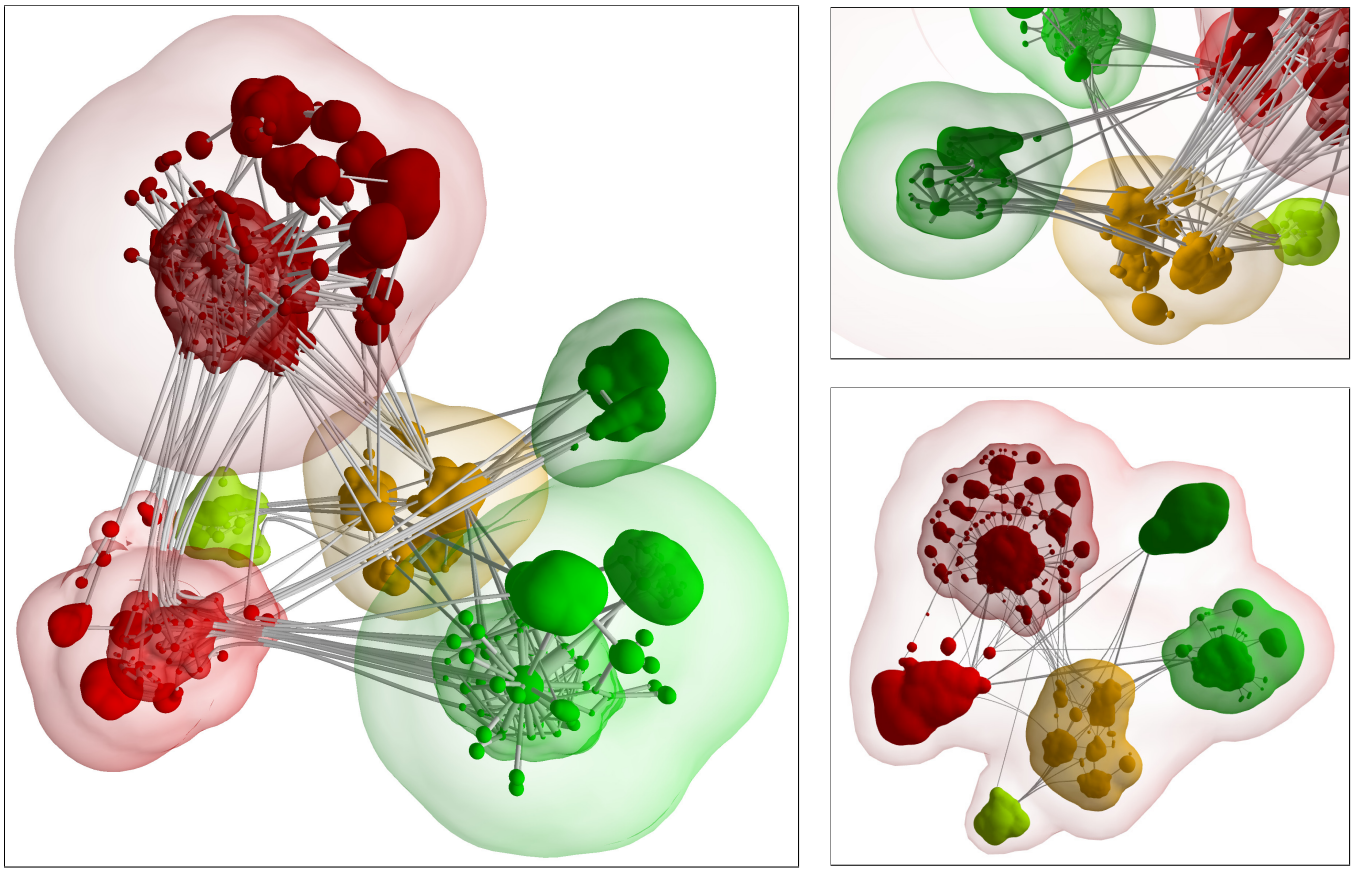


Figure 9: Views of a real world graph in $\mathbb{R}^3$ (left, upper right) and $\mathbb{R}^2$ (lower right) representing relations within a software system. The graph contains 1539 vertices, 1847 edges, and 126 clusters. The computation of the implicit surfaces end edge bundles required less than one minute.