Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Institute of Scientific Computing

Computer Graphics Group

CS Technical Report #341, April 18, 2000

A Hierarchical Visual Clustering Method Using Implicit Surfaces

T. C. Sprenger, R. Brunella, M. H. Gross

Computer Science Department
ETH Zurich, Switzerland
e-mail: {sprenger, grossm}@inf.ethz.ch
http://graphics.ethz.ch/

H-BLOB: A Hierarchical Visual Clustering Method Using Implicit Surfaces

T. C. Sprenger, R. Brunella, M. H. Gross

Department of Computer Science
Swiss Federal Institute of Technology (ETH)
Zurich, Switzerland

ABSTRACT

In this paper, we present a new hierarchical clustering and visualization algorithm called H-BLOB, which groups and visualizes cluster hierarchies at multiple levels-of-detail. Our method is fundamentally different to conventional clustering algorithms, such as C-means, K-means, or linkage methods that are primarily designed to partition a collection of objects into subsets sharing similar attributes. These approaches usually lack an efficient level-of-detail strategy that breaks down the visual complexity of very large datasets for visualization. In contrast, our method combines grouping and visualization in a two stage process constructing a hierarchical setting. In the first stage a cluster tree is computed making use of an edge contraction operator. Exploiting the inherent hierarchical structure of this tree, a second stage visualizes the clusters by computing a hierarchy of implicit surfaces. We believe that H-BLOB is especially suited for the visualization of very large datasets and for visual decision making in information visualization. The versatility of the algorithm is demonstrated using examples from visual data mining.

keywords: clustering, categorization, partitioning, information visualization, non-linear dimensionality reduction, physics-based graph layout, cluster visualization, multidimensional information visualization.

1 INTRODUCTION

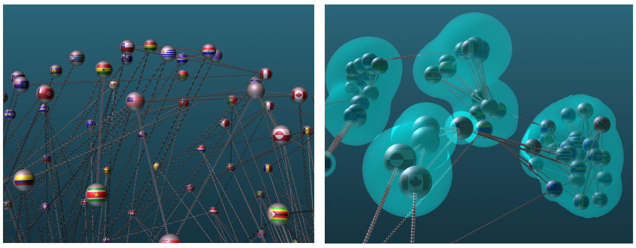
The term *clustering* refers to the process of grouping similar objects, where similarity is captured by a metric function [28], [29].

Clustering methods have been a hot topic in different research fields such as: statistics, pattern recognition, machine learning, etc. Because of the constantly increasing size of datasets over the last years, clustering also has advanced to a key technology in the area of *information visualization* and *data mining*. In fact, with the use of today's technology for data generation and collection, typical datasets have grown by magnitudes. Since the human cognitive system is limited to recognize only a very small number of objects at once (around 7 objects) as well as due to performance restrictions of today's graphics hardware we are forced to the use an efficient level-of-detail strategy. Consequently, literature describes various interesting data clustering approaches including their efficient and refined implementations [8], [9], [13], [17], [18], [19].

Because our main interest lies in visualizing clusters, we focus on the problem of clustering large data sets in *coordinate space* [12], also referred to as the *Euclidian space*, in which data objects can be represented as vectors $v \in R^n$. Unlike data sets in a *distance space* [12], also referred to as the *data domain* or the *arbitrary metric space*, the vector representation gives access to various efficiently implemented vector operations (e.g. addition, multiplication, dot-product, etc.), which enables one to calculate simplified representations of complex data subregions at interactive rates. Similar operations are not defined in distance space. The only possible operation is the computation of a distance function between two data objects, thus rendering the problem of clustering much more complex.

Since many problems in information visualization are located in distance space, and thus non-accessible for our methods, a projection from distance space into coordinate space has to be defined. Such a projection operator maps each data object from distance space to an n-dimensional vector in coordinate space while preserving relative distances between objects. Thereafter, vector-based clustering methods may be applied and their results can be visualized in 2D or 3D space.

This approach entails an additional advantage. Once the projection operator has been applied, the objects have become data-independent, i.e. the clustering algorithm operating on those objects is highly reusable for a large variety of data clustering tasks.



a) b)

Figure 1: Clustering of a subset of objects performed with BLOBS. a) Initial object layout b) Clustered configuration with enclosing BLOB surface.

There exist several techniques for topology-preserving transformations [24]. One of them is called *multidimensional scaling* (MDS) [27]. Other widely spread methods are employing with neural networks, namely with topology-preserving *Kohonen networks* [25], [26], which belong to the group of *self-organizing features maps* (SOM). As a third technique *spring-embedding systems* (SES) perform the desired transformation by running a physics-based simulation process [22], [23].

Our clustering research activities take place in the context of the IVORY project, where we develop a JAVA-based framework for physics-based visualization and analysis of multidimensional data relations [5], [6]. The system is based on quantifying the similarity of related objects, which governs the parameters of a *spring-embedding system*. Since the spring stiffnesses correspond to the computed similarity measures, the system will converge into an energy minimum, which reveals multidimensional relations and adjacencies in terms of spatial neighborhood. In our research work, IVORY serves as a versatile information visualization environment to explore visual metaphors and advanced interaction paradigms.

In order to simplify the geometry and topology of complex object setups, IVORY already provides a set of clustering algorithms for postprocessing. In contrast to many other cluster-based

Reviewer's remark: We added an optional appendix page to the paper in order to provide additional information to set this work into the proper context. If paper length is an issue the appendix page could be dropped for a final version.

systems, IVORY not only calculates clustered object layouts including corresponding one-level partitions (as a group of cluttered single objects) but also computes an enfolding surface (ellipsoids, BLOBS (implicit surfaces), etc.) for each cluster [5], [6]. Aiming at a reduction of complexity, such a surface can replace a large group of single objects in a higher level of representation. Without losing significant visual information, the scene can drastically diminish in complexity. At the same time, the visual distinctness increases.

In this paper we introduce the concept of H-BLOB clustering. Our new technique discovers and visualizes clusters by a two-stage procedure. During the first stage, an agglomerative hierarchical algorithm computes a cluster tree, partitioning data objects into a nested sequence of subsets. This is what we call the *analytical clustering step*. In a second stage, the intrinsic visualization takes place. We compute a single enclosing shape for each cluster which approximates the outline of the included data objects as closely as possible. For the visualization we propose a new technique called H-BLOBS, which is a direct improvement to the BLOB clustering algorithm presented in [5].

The remainder of the paper is organized as follows. In Section 2, we discuss related work on clustering and some of our initial approaches. In Section 3, we present the technique we use for fast analytical clustering and introduce the H-BLOB algorithm dedicated to visualize cluster hierarchies using implicit surfaces. The paper closes with Section 4 describing the implementation issues and its versatility on the basis of a real world example.

2 RELATED WORK AND FUNDAMENTAL APPROACHES

Clustering algorithms can be roughly divided into two categories: *partitioning* and *hierarchical* methods. In the following two subsections we present a variety of widely used partitioning, respectively hierarchical clustering algorithms, followed by a description of different advanced cluster visualization techniques.

The following list is far from being complete, but it should point out the main clustering techniques, most of today's clustering algorithm are base upon. Mainly, this section conduces to set our work into context and better understand our approach.

2.1 Partitioning Methods

Partitioning cluster methods (PCM) attempt to analytically subdivide a set of data objects into a certain number of clusters, whereupon they assume that clusters are of hyper-ellipsoidal shape and of similar size. Like other centroid-based techniques they generally fail, if clusters differ significantly in shape or size. We will have a closer look at two representative algorithms and their qualities.

C-Means

The basic idea of the *C-means method* is to join an object obj_i to a cluster $clust_j$ if the distance between the position $\boldsymbol{x}_i$ of the data object obj_i and the center $\boldsymbol{c}_j$ of the cluster $clust_j$ is less than a threshold value δ:

$$abs(\boldsymbol{x}_i - \boldsymbol{c}_j) \leq \delta \tag{1}$$

The center position $\boldsymbol{c}_j$ of cluster $clust_j$ is defined by the arithmetic average of the positions of all data objects $\boldsymbol{x}_i$ enclosed by cluster $clust_j$

$$c_j = \frac{1}{N} \cdot \sum_{i=1}^{N} x_i \tag{2}$$

where N designates the number of data objects within the current cluster.

The C-means algorithm iterates over all data objects obj_i and verifies for each object obj_i if there exists a cluster $clust_j$ the center $\boldsymbol{c}_j$ of which is closer to $\boldsymbol{x}_i$ than δ. If there are such clusters the object will be added to the cluster that is closest to the object. Otherwise a new cluster is generated with the object $\boldsymbol{x}_i$ as its only member. After assigning the object to the cluster's center position will be updated, i.e. the center will shift.

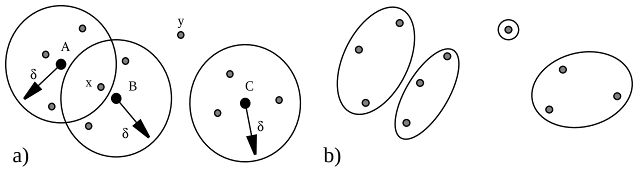


Figure 2: a) Partitioning using C-means method with threshold δ, where the assignment of object x is undetermined. Object y, on the other hand, could not be assigned to any existing cluster. Therefore, it generates a new one. b) Completely clustered scene.

A major disadvantage of the C-means method is the user defined selection of the cluster threshold value δ. Eventually, the determination of a proper value for δ could be very difficult. With too large a value clusters will contain objects which do not correspond. On the other hand, too small a value will result in clusters each holding only one single object. Another drawback is the sensitivity of the algorithm to the order of traversal of given objects. In particular, the choice of the starting object has a great influence on the resulting cluster distribution.

The cost of the C-means algorithm is of order $O(n^2)$ being defined by the worst case scenario, with each object located in its own cluster. But due to the very simple operations the C-means method relies on, it is very fast in general.

K-Means

K-means belongs to the class of iterative clustering techniques. Choosing the *K-means method* we have to preselect the number k of clusters, the algorithm would generate.

First k initial cluster centers are defined. An object obj_i is assigned to the cluster $clust_j$ when its center $\boldsymbol{c}_j$ is closest to the object position $\boldsymbol{x}_i$. In such a way, all objects are associated to exactly one cluster. At the beginning of the next iteration, the cluster centers $\boldsymbol{c}_j$ of all k clusters are updated to the arithmetical average of all positions $\boldsymbol{x}_i$ of associated objects. Thereafter, another assignment round starts using the recently computed cluster centers. The iteration loop stops if all cluster centers have converged into a stable position.

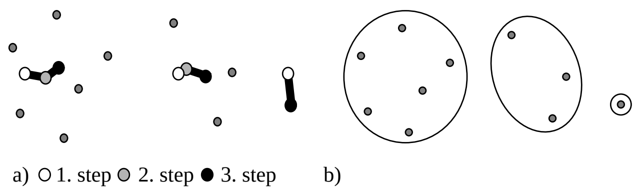


Figure 3: The same scene as in fig. 2 clustered with the K-means algorithm a) The iteration steps for the 3 cluster centroids. b) Resulting clustered layout.

The K-means method poses a problem concerning the selection of the initial positioning of the k Clusters. A unlucky choice could have great influence on the resulting object clustering.

K-means' iterative behavior and the apriori unknown number of iterations makes the cost estimation more difficult than for the C-means algorithm. In each step, the algorithm calculates the distances between all *n* object and the *k* cluster centers, i.e. calculates *nk* distances. Since *k* is constant, the costs are of order $O(n)$ per iteration step.

2.2 Hierarchical Methods

Hierarchical clustering methods (HCM) are commonly used in the area of information visualization and data mining. In contrast to partitional clustering methods, that subdivide a set of objects into a certain number of clusters, hierarchical clustering generates a nested sequence of partitions. We call this a *cluster tree* (as shown in fig. 4).

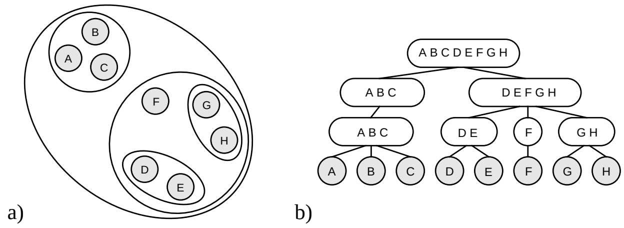


Figure 4: a) Probable object arrangement with 8 objects. b) Corresponding cluster tree with 4 levels generated by an agglomerative, hierarchical clustering algorithm

An *agglomerative hierarchical clustering* algorithm starts with *n* atomic clusters, each containing exactly one object. At each step, the algorithm merges the two most similar[1] clusters and thus decreases the total number of clusters by one. These steps recur until only one single cluster, containing all objects, remains. Any two clusters generated by such a procedure are either nested or disjoint. In contrast, *divisive hierarchical clustering* reverses the process by starting with a single cluster holding all objects and subdividing it into smaller sets [8].

Many variants of agglomerative hierarchical clustering methods are known, mainly differing in the definition of the metric applied in updating the similarity between existing and merged clusters.

Along with the incremental algorithms mentioned above, there is a group of non-incremental clustering methods (e.g. CLUSTER/S [20]). The discussion of those algorithms is beyond the scope of this paper, and their methods are not considered in the following.

In the remainer of the section we shall discuss two different hierarchical clustering methods: the *single linkage method* and the *complete linkage method*. For an in-depth description we refer to [7].

Single Linkage Method

Another straightforward and quick clustering technique is called *single linkage method* (SLM) or *nearest neighbor technique*. For this algorithm we define the distance between two clusters as the minimal spacing between two arbitrary objects, each located in two different clusters. Assume that d_{ij} is the distance between object obj_i from cluster $clust_i$ and object obj_j from cluster $clust_j$. Then, the distance D_{ij} between clusters $clust_i$ and $clust_j$ is defined as

$$D_{ij} = min(d_{ij}) \quad . \tag{3}$$

That means we measure distances between two clusters as the distance of the closest pair of objects each belonging to a different cluster. The SLM synthesizes clusters analogous to the general description found at the beginning of this section.

[1] In the current context similarity of two objects is defined by the inverse of their distance. Thus the algorithm merges the two closest clusters in each step.

A problem of SLM is the algorithm's tendence to generously accept object chains as clusters. Assume we have an object configuration like the one shown in fig. 5. The SLM would string objects between A and B to a chain. Thus, objects A and B will be assigned to the same cluster. SLM generates three clusters (drawn with a solid line). Building only two clusters (shown with a dotted line) would be a superior solution.

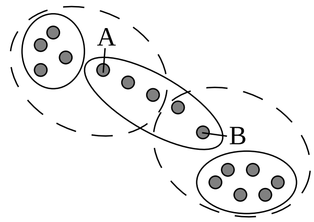


Figure 5: Generation of chains applying the single linkage method

Unlike centroid-based algorithms, this method could discover clusters of arbitrary shape and different size. Unfortunately, the procedure is highly susceptible to noise and outliners.

To build up the cluster tree, the single linkage method has to compute the pairwise distance between every two objects, i.e. supposed we have *n* objects, we have to perform $n \cdot (n-1)/2$ distance evaluations per iteration, which dearly is of order $O(n^3)$ over all *n* iteration steps.

Complete Linkage Method

Another clustering method, the *complete linkage method* (CLM), takes into account the chain formation and defines the distance between two clusters D_{ij} as the maximal distance between two of their objects

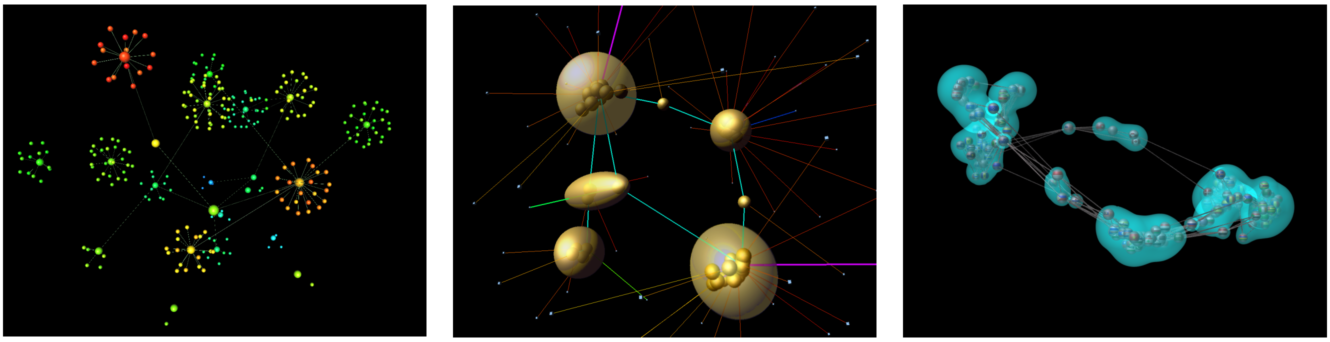
$$D_{ij} = max(d_{ij}) \tag{4}$$

Supposed we run the CLM on an object topology that already contains two shorter cluster chains, the distance between the two clusters is now defined by the two furthest away objects not located in the same cluster. This is equal to the distance of the outermost object on the one side of a chain and the outermost object on the other side of the other chain. Thus, chain formation is suppressed.

As mentioned at the beginning of this section, there are many other well known clustering algorithms, i.e. BIRCH [17], which is basically an extension of the K-means clustering, but adequately addresses the problem of large datasets. CURE [18] remedies the drawback of single centroid representation by taking advantage of a multi-centroid representation of clusters. Hence this algorithm is more robust to outliners and identifies clusters varying in size and having non-spherical shapes. A recent approach is called CHAMELEON [19], a hierarchical clustering algorithm that measures inter-cluster similarity based on a dynamic model. In addition to other algorithms, CHAMELEON clustering is based not only on vicinity of objects but also considers corresponding connectivity information. This combination results in a robust handling of data that consists of clusters being of different shape, size or density.

2.3 Cluster Visualization Methods

There is quite large a number of algorithms and systems treating the subject of cluster visualization. Practically all of them take the problem of cluster visualization simply as a layout problem, thus focusing on optimizing the computation and spatial grouping of crowds of single data objects. The visualization then is limited to drawing just a simple shape (dot, icon, glyph, etc.) for each data



a) b) c)

Figure 6: Different techniques to visualize clusters of data objects. a) cluster represented by a cluttered group of single objects b) visualization with ellipsoidal surfaces wrapped around clusters c) objects visually combined by a BLOB surface.

object (shown in fig. 6a). Thus, the actual visual clustering process is rather done by the user's perceptual system than by the visualization system itself.

There are two reasons to go a step further: first today's graphics hardware, though current progress in this area is tremendous, is not yet ready for the data volumes we would like to address with present data management systems (i.e. data warehouses). Second, the user's perceptual system should be relieved of gathering single points to a cluster object. In order to speed up the decision making process and to increase the decision's quality, cluster visualization has to take the step to the next higher level of visual representation.

Only a few approaches make an effort in this direction. Some of the systems attempt to break down complexity by running a preclustering algorithm on the initial dataset. Afterwards the system confines itself to displaying only objects on a chosen clustering level, where clusters are represented by a simple shape at the position of their centroids. Doing so, we lose most of the information contained in a cluster. Only the cluster's position is visible to the user. Information about the internal object distribution, including size, orientation and variation is visually not available to the user.

Initial work about a more powerful visualization method is reported in [30], where wrapping hyperspheres accomplish the clustering of data objects. Furthermore, some of the authors of this paper proposed a PCA-based technique in [4] where the basic idea was to wrap ellipsoids around each object group whose shape is controlled by the principal components of the respective cluster (shown in fig. 6b). In either approach restriction to a quadric surface representation of the clustering hull represents an unnecessary restriction. The internal object distribution is only rough an approximation, as well in size as in orientation. This drawback gets addressed by an algorithm called BLOB-clustering [4], the fundamental idea of which is to use *blob functions* combined with a m*arching cube* [3] algorithm to represent the enfolding cluster surface (see fig. 6c). The generated shape represents the distribution of the included data objects in the best possible manner.

However, all of the cluster visualization methods mentioned above are limited to work only based on *partitioning clustering algorithms*. Non of them takes advantage of the hierarchical information cluster structures inherently contain. Therefore, we propose a new simple and fast clustering technique that has its strength in the visualization of hierarchical clustering structures, say *cluster trees*.

3 H-BLOB: HIERARCHICAL CLUSTER VISUALIZATION USING ISOSURFACES

The *H-BLOB (Hierarchical BLOB) algorithm* is considered to be a direct derivative of the BLOB clustering method, extended by the capability to handle hierarchical settings. In fact, it is a combination of techniques and algorithms described in preceding sections, each one applied on a preferable subtask corresponding to their strengths.

The algorithm can be split into two stages, starting with an analytical clustering process building up a cluster tree, which is followed by the hierarchical cluster surface computation in combination with the visualization process.

3.1 Stage I: Edge Collapse Clustering

Inspired by the persuasive idea of the *edge collapsing algorithm* presented in [2], we propose a new simple and efficient clustering method, called *edge collapse clustering* (ECC).

The algorithm we present, belongs to the category of *agglomerative hierarchical clustering* methods. Thus, the general structure is very similar to the methods presented in Section 2.2.

In contrast to the linkage methods the ECC bases on centroids; hence, it only works in *coordinate space*. We define the distance D_{ij} between two clusters $clust_i$ and $clust_j$ as the distance between their centroids $\boldsymbol{c}_i$ and $\boldsymbol{c}_j$

$$D_{ij} = abs(\boldsymbol{c}_i - \boldsymbol{c}_j) \ . \tag{5}$$

The process of cluster merging works analogous to the process shown in Section 2.2, but with the following extension:

All clusters $clust_i$ obtain a weight w_i corresponding to the number of objects contained in $clust_i$. The weight w_i is initialized with a value of one. With each iteration, the algorithm merges the two closest clusters, i.e. the pair of clusters with minimal distance D_{ij}, into a new one, called $clust_{new}$ with centroid $\boldsymbol{c}_{new}$. At the same time, the parameters of the new cluster are updated corresponding to the formulas below:

$$\boldsymbol{c}_{new} = \frac{\boldsymbol{c}_i w_i + \boldsymbol{c}_j w_j}{w_i + w_j} \tag{6}$$

$$w_{new} = w_i + w_j \tag{7}$$

If the two clusters are of different weight, the new cluster will be located closer to the heavier, i.e. larger cluster, which is desirable in praxis.

Fig. 7 illustrates the algorithm by means of an example with 5 objects spread on a plane. Each iteration step is shown on a separate line, with the actual object arrangement in the left half and the current cluster tree on the opposite side. Starting with 5 single objects, the ECC algorithm merges them into a single cluster after the same number of iteration steps. The red line, highlights the edge to be collapsed next.

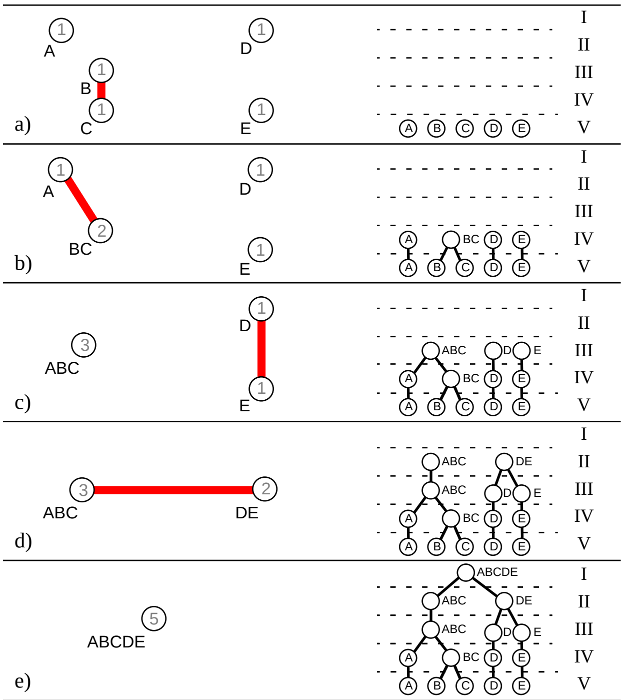


Figure 7: a) - e) Progressive edge-collapse algorithm. Red line indicates edge to be collapsed next. Current cluster tree levels (I-V) are shown on the righthand side.

Since each cluster is defined by its centroid only and as the distance metric depends only on the centroid's coordinates, every two clusters are virtually interconnected with exactly one edge of length D_{ij}. Consequently, ECC takes advantage of the inherent hierarchical structure of a cluster tree. The computational complexity for each iteration step is defined by the corresponding number of clusters. This is an advantage compared to the linkage algorithms, which always operate on the initial set of all single objects. Hence, the ECC algorithm is computational less complex than linkage methods.

The disadvantages concerning the fragile user-driven parameter preselection of the C- and K-means methods do not apply for ECC. Although this technique is partly based on centroids, it is more stable with respect to unconstrained shapes and different cluster sizes than C- and K-means. The effect of chain formation does not occur for ECC.

Unfortunately, the ECC is still in the same polynomial order as the linkage techniques. It also preforms n iterations steps and computes in each of the steps $n \cdot (n-1)/2$ distances. Since ECC computes distances based on centroids we get a triangular cost scheme over all iterations, which results in an complexity of order $O\left(\frac{1}{6}n^3 - \frac{1}{6}n\right)$ regarding the number of computed distances.

3.2 Stage II: Cluster Tree Visualization

The cluster tree generated as a result of the first stage must now be visualized, Each hierarchy level should be handled separately, i.e. we compute a separate surrounding surface for each cluster at a specific hierarchy level.

As a basic idea we devote resources to the BLOB algorithm described in [5]. The fundamental idea of BLOB clustering is to give each object a spatial extension by attaching a spherical *primitive* to its center. In general a *primitive* is a working model comprising a parameterized oriented shape and a corresponding 3D field function $f_i(x, y, z)$. Primitives and their parameterization will be explained in more detail in the next section.

To compute a BLOB surface, we superimpose all field functions $f_i(x, y, z)$ in space and accordingly run a *marching cube algorithm* [3] to extract the implicit surface at a given *isovalue*. The subsequent sections explain how we extend this algorithm in order to handle hierarchical cluster structures efficiently.

Visualization using BLOBS

As a straightforward approach to visualize a single cluster on a given cluster level, we could assume a scenario where a primitive is attached to each of the cluster's objects. Supposed we choose a skillful parameterization of those primitives, we could accomplish an isosurface, that fully encloses all objects and the visualization problem would be superficially solved.

Even if this approach results in fair visual results, it has a tremendous handicap. For very large clusters holding a huge number of single objects the computational cost rises exponentially. That effect occurs because in order to perform an isosurface extraction we have to evaluate the superimposed field at given points in space which involves the evaluation of the field equation for every single primitive. The problem could be eased if we find a way to limit the number of primitives during visualization.

We consider the cluster tree shown in fig. 8, subdivided into 3 hierarchical levels. The topmost cluster on level I contains all 5 objects (ABCDE). If we intend to visualize this cluster, we have to take into account five different primitives – one for each object.

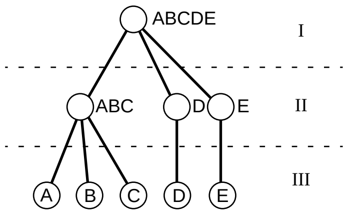


Figure 8: Cluster tree with three levels. It is a condensed view on the corresponding tree shown in fig. 7e without displaying level II and IV.

To limit the number of primitives we propose the following approach: instead of attaching primitives to every single object, we just consider the objects one level below the level of interest. Thus, in order to visualize the cluster in level I we attach primitives to the level II cluster objects, i.e. to the clusters (ABC), (D) and (E). Or, if we aim to visualize clusters of level II, we utilize cluster objects from level III and so forth.

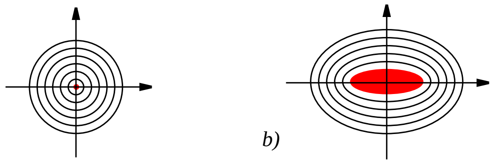
To provide for satisfactory results, we need to extend the characteristics of the primitives used, which – in the original BLOB paper [5] – were restricted to be of radial symmetric shape. This is due to the fact that in contrast to the previous BLOB clustering algorithm primitives now have to account for the properties of a whole object set rather than of only one single object. We suggest the extension of our concept of a primitive to an elliptical feature, the so called *ellipsoidal primitive*. The following sections will give a more exact definition.

Extension to Ellipsoidal Primitives

Ellipsoidal primitives are a direct extension to the common primitives determined in [5]. The characteristics of an ellipsoidal primitive is specified by an ellipsoidal shape and the field function f_i. For the definition of the shape and the computation of its size, orientation and position we refer to [4]. The definition of f_i is

$$f_i(x, y, z) = \begin{cases} b_i & \text{if } (x, y, z) \text{ lies inside ellipsoid} \\ b_i \cdot e^{-a_i d_i(x, y, z)^2} & \text{else} \end{cases} \tag{8}$$

where $d_i(x, y, z)$ is the distance to ellipsoidal surface, b_i defines the maximal magnitude of the function inside the ellipsoid, and a_i influences the descent of the field function.



a) b)

Figure 9: Isolines of a) a spherical, symmetric primitive and b) a new ellipsoidal primitive.

Fig. 9 compares the fields of a spherical symmetric primitive to the field of a new ellipsoidal primitive defined by eqn. (8) on the basis of their isolines. Inside the red area the field has a value of b_i.

The field f_i of a single ellipsoidal primitive could be described as follows: for all points inside the ellipsoid the value of the field is uniformly b_i. Starting at the surface of the ellipsoid the field descents exponentially and monotonously as a function of the distance to the surface.

Computation of Ellipsoidal Gaussian Fields

An ellipsoid is defined by its scaling matrix *S*, its rotation matrix *R* and its center $\boldsymbol{m}$. From the diagonal elements of the scaling matrix result the three half axes H_a, H_b and H_c.

Transforming the ellipsoid into the origin will simplify subsequent formulas. In order to compute the value of the field function f_i at a point $\boldsymbol{p} = (x, y, z)$ from eqn. (8), the coordinates of $\boldsymbol{p}$ have to be transformed: first, $\boldsymbol{p}$ is translated by the negative values of vector $\boldsymbol{m}$ according to

$$\boldsymbol{p}' = \boldsymbol{p} - \boldsymbol{m}\,. \tag{9}$$

Then, $\boldsymbol{p}$ is rotated by the inverse rotation matrix R:

$$\boldsymbol{p}'' = R^{-1} \cdot \boldsymbol{p}' = (x'', y'', z'')^T \tag{10}$$

To gather the distance between the transformed point $\boldsymbol{p}''$ and the surface of the ellipsoid, it is necessary to intersect the connecting line between the center of the ellipsoid – which is equal to the origin – and the point $\boldsymbol{p}''$ with the ellipsoidal surface. To this aim the line $\overline{\boldsymbol{Op}''}$ is parametrized with t running from 0 to 1.

$$\boldsymbol{p}_t(t) = (x_t, y_t, z_t)^T = (t \cdot x'', t \cdot y'', t \cdot z'')^T \tag{11}$$

A point $\boldsymbol{p}_t$ is located on the surface of the ellipsoid, if the ellipsoidal equation evaluates to 1:

$$\frac{x_t^{\,2}}{H_a} + \frac{y_t^{\,2}}{H_b} + \frac{z_t^{\,2}}{H_c} = 1 \tag{12}$$

Substituting eqn. (11) into eqn. (12) yields for the intersection point t_s:

$$t_s = \sqrt{\frac{H_a^2 \cdot H_b^2 \cdot H_c^2}{x''^2 \cdot H_b^2 \cdot H_c^2 + y''^2 \cdot H_a^2 \cdot H_c^2 + z''^2 \cdot H_a^2 \cdot H_b^2}} \tag{13}$$

If $t_s > 1$, then the point lies within the ellipsoid.

With it f_i could be computed using transformed coordinates:

$$f_i(x'', y'', z'') = \begin{cases} b_i & ,t_s > 1 \\ b_i \cdot e^{-a_i \cdot (1 - t_s)^2 \cdot (x''^2 + y''^2 + z''^2)} & ,t_s \le 1 \end{cases} \tag{14}$$

Parameter Definition for Ellipsoidal Primitives

The *ellipsoidal primitives* contain the two parameters a_i and b_i, which control the descent and magnitude of the corresponding field function. These two parameters should be determined automatically, because a configuration by the user may be longsome and instable. Whenever possible, the algorithm should disburden the user from such decisions.

The simplest approach would be a static setting for these two parameters. Unfortunately, this idea is not acceptable because the visualized clusters vary too much in both scale and position. Thus, it is impossible to find values that delivering satisfactory results under all circumstances. The parameters have to set in context with the underlying ellipsoid. We will discuss two possible approaches solving this problem:

1. The heavier a cluster is, i.e. the more objects it contains, the larger becomes the value of the magnitude b_i of the ellipsoid primitive's field function.
2. The larger the maximum extension of the ellipsoid is, the weaker becomes the descent a_i of the ellipsoid primitive's field function.

Experiments have shown, rule one can lead to very big BLOB surfaces, e.g. if the object distribution in space is dense. Hence, this rule was dropped and a fixed value is assigned to b_i (e.g. b_i=1.0).

The second rule on the other hand is considered to provide an relevant visual feedback. The parameter a_i is defined as

$$a_i = \frac{a_0}{ellipsoid's\ dimensions} \tag{15}$$

where the value for the constant factor a_0 must be determined experimentally, yet.

Determination of Isovalues to ensure connected BLOB-Surfaces

According to [5] a BLOB's shape is strongly influenced by the corresponding isovalue $c > 0$. The smaller this value, the larger the BLOB's extension will get. In order to ensure that a BLOB encloses all its objects the correct choice of c is crucial. In this section, heuristics for the automatic determination of isovalues is presented.

Take the example of fig. 10 where an enclosing BLOB surface for three objects A, B and C has to be computed. The indicated number on the connecting edges illustrates the minimal value of the superimposed field along the edge. In order to assure as tight a BLOB as possible we have to look for the largest isovalue which still guarantees that the BLOB does not break apart.

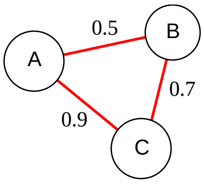


Figure 10: Three objects for which an enclosing tight BLOB surface has to be found.

Fig. 11 shows three possible cases for the choice of an isovalue. On the left hand side, the chosen value results in the illustrated split-up into two subclusters because $c = 0.8$ is bigger than the minimal field value an edges AB and BC. On the right hand side, too small an isovalue does not provide for a distinctive shape. The case illustrated in the middle seems ideal. Choosing $c = 0.6$ – bigger than the minimum on edge AB but smaller than the minimal value on BC – results in a tight single BLOB surface enclosing all objects.

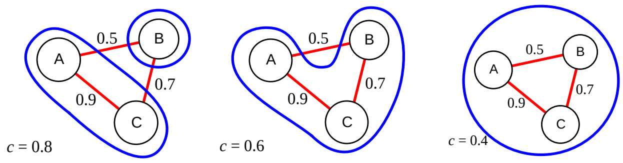


Figure 11: left: iso-value too big, BLOB breaks apart
middle: optimal iso-value, tight BLOB enclosing all objects
right: iso-value too small, non-distinctive shape

This example shows how to find an ideal isovalue: look for the biggest value that still guarantees for a single enclosing surface. This is equivalent to choosing a value such that all objects are connected by edges with minimal field value bigger or equal to the isovalue.

There are two problems in this approach: first, graph theory shows that it is very expensive to find a minimal spanning tree, at least if cluster sizes approach several hundred objects. Second, finding the minimal field value on interconnecting lines is expensive too, as it is impossible to find an analytic solution for arbitrarily superimposed fields. In the remainder of the section, we present an approach which in most cases yields suited isovalues.

Fig. 12 shows a constellation of several objects of a cluster for which an enclosing BLOB surface has to be found. The red dot marks the center of the cluster. Intuitively, objects close to cluster center will not cause problems. In contrast thereof, it is troublesome to account for outliers – objects which are far apart from the cluster's center. Instead of looking for a minimal spanning tree for all of the cluster's objects we concentrate on the outliers. Therefore, we look for the minimal field value on the interconnecting lines between the outlier and the cluster center. Fig. 12 shows these lines highlighted in red. The smallest value found is regarded as a good approximation to the ideal isovalue.

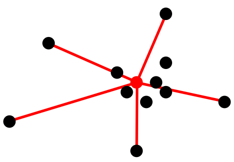


Figure 12: Objects of a cluster with so-called outlier objects. The interconnecting lines between outliers and the cluster center are marked in red.

We are left with the problem of finding the minimal field value on the lines between outliers and the cluster center. To this aim, we employ a Newton iteration scheme in order to find the zero crossings of the first derivative of the superimposed field function with regard to parametrization t of the interconnecting line

$$f'(t) = 0\,. \tag{16}$$

The corresponding Newton iteration step is given by

$$t_{n+1} = t_n - \frac{f'(t)}{f''(t)}\,. \tag{17}$$

As it is hardly possible to find symbolic expressions for the first and second derivative of the field function f, they are approximated in terms of central differences as follows:

$$\begin{aligned} f'(t) &\approx \frac{f(t+\Delta t) - f(t-\Delta t)}{2} \\ f''(t) &\approx \frac{f'(t+\Delta t) - f'(t-\Delta t)}{2} \\ &= \frac{2f(t+2\Delta t) - 2f(t) + 2f(t-2\Delta t)}{4} \end{aligned} \tag{18}$$

As the reader may have noticed, this procedure is not guaranteed to find the global minimum but is highly dependent on the choice of a favorable initial value t_0. In order to find a good value for t_0, we sample the value of the field function on equidistant points on the interconnecting line and choose t_0 to be the smallest value found during the sampling procedure. As a matter of fact, the outlined procedure still does not provide for finding the global minimum. However, practice has shown, that it yields suitable isovalues for non-pathological cases. For clusters of less than five objects the minimal spanning tree is computed which guarantees for the optimal isovalue.

4 IMPLEMENTATION AND RESULTS

This section documents a concrete implementation of the H-BLOB algorithm in the context of our information visualization research project, called IVORY. Following, on the basis of two examples we illustrate the visual performance and versatility of our approach.

4.1 Implementation

The algorithm has been fully implemented as a class library in Java2. For the domain of 3D visualization we apply Java3D in the version 1.1.2. All computational work is done on a standard PC completed with a hardware accelerated graphics subsystem (Open GL). Even for more complex examples we still get interactive frame rates.

Concerning an implementation of the H-BLOB algorithm there are two main issues. The first one affects the data structure used for the edge collapse clustering. Since this stage of the algorithm makes heavy use of point-to-point distance calculations and cluster merging, together with the higher order characteristic of the problem, makes a good choice difficult. Employing standard data structures quickly leads to a performance bottleneck, mostly because of memory shortage. Some promising work addressing this type of problems could be found in [1]

The second issue is about the isosurface extraction. In spite of the multi-resolution approach it remains the most time consuming part of the algorithm. Implicit surfaces may provide very nice shapes, but are computational very expensive. There are many sources available to this topic, but for our prototype implementation our choice was [3].

4.2 Small World Example

This first and small example illustrates the basic properties of the H-BLOB clustering algorithm. The scene consists of 5 single objects each represented by a colored sphere. We present two snap-shots of the cluster tree buildup sequence including the corresponding implicit cluster surfaces generated by the H-BLOB algorithm.

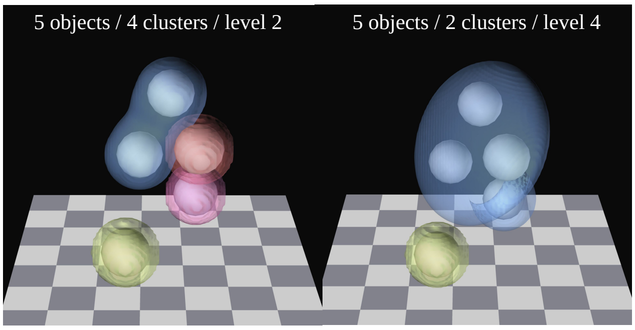


Figure 13: Small example showing the clustering process by means of 5 simple objects. Snapshot with 4 and 2 clusters are shown. Level indicates the hierarchy level in respect to the cluster tree.

4.3 Document Retrieval Visualization

This example is a from a real document retrieval research project. We applied our new technique to a hit list (result list) originate from an intranet document query. The number of single objects is 100. For the clustering stage a maximum of 20 clusters has been defined. From one picture to the next we respectively merge 50% of the clusters, what results in 6 hierarchy levels with 20, 10, 5, 3, 2, and 1 clusters. We show 4 selected images from this session.

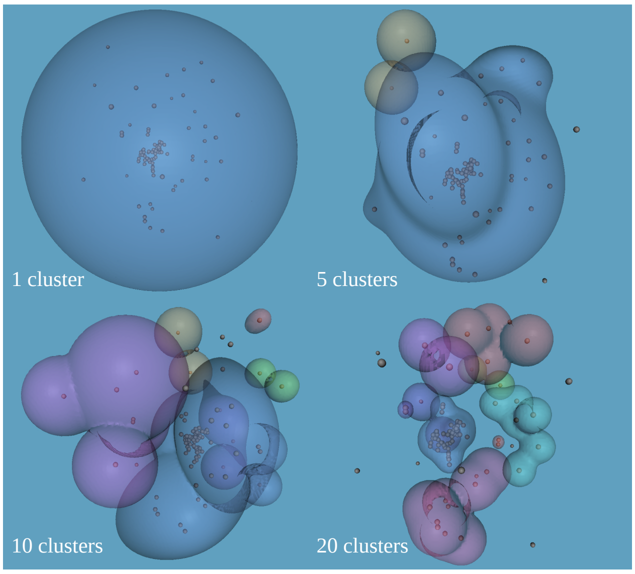


Figure 14: Document Retrieval Visualization. Cluster hierarchies are shown with 20, 10, 5 and 1 cluster.

5 CONCLUSION

The main contributions of this paper is a new hierarchical clustering algorithm called H-BLOB, which provides an efficient level-of-detail strategy and is consequently capable to cluster and visualize very large and complex data volumes. The algorithm is subdivided into two stages: Firstly, a simple and fast clustering strategy – based on edge collapsing – computes a cluster hierarchy. Secondly, improving this hierarchical structure, the next stage visualizes the clusters with nested implicit shapes. The key concept is an efficient multi-resolution setup, breaking down the structural and visual complexity of scenes. We have shown the algorithm's versatility by experimental results, demonstrating H-BLOB's capability to simplify and enhance the feasibility of cluster visualization.

ACKNOWLEDGMENT

This research has been made possible by the Advanced Engineering Center (AEC) of the UBS, Basel, Switzerland. Many thanks to Martin Roth and Andreas Hubeli for their extraordinary efforts in text editing and proofreading.

REFERENCES

[1] D. Eppstein. “Dynamic Euclidean minimum spanning trees and extrema of binary functions.”, *Discrete & Computational Geometry 13(1):111 122*, January 1995.

[2] H. Hoppe. “Progressive meshes.” *In Computer Graphics (SIGGRAPH 1996 Proceedings)*, pp. 99-108, Aug. 1996.

[3] J. Bloomenthal. "An Implicit Surface Polygonizer." *In P. Heckbert, editor, Graphics Gems IV*, Academic Press, Boston, pp. 324-349, 1994.

[4] T. C. Sprenger, M. H. Gross, A. Eggenberger, M. Kaufmann. “A Framework for Physically-Based Information Visualization.” *Proceedings of Eurographics Workshop on Visualization ’97* (Boulogne sur Mer, France, April 28-30, 1997), pp. 77-86, 1997.

[5] M. H. Gross, T. C. Sprenger, J. Finger. “Visualizing Information on a Sphere.” *Proceedings of IEEE Information Visualization ’97* (Phoenix AZ, USA, 19-24 October 1997), pp. 11-16, 1997.

[6] T. C. Sprenger, M. H. Gross, D. Bielser, T. Strasser. “IVORY - An Object-Oriented Framework for Physics-Based Information Visualization in Java.” *Proceedings of IEEE Information Visualization ’98* (Research Triangle Park, NC, USA, October 19-20, 1998), pp. 79-86, 1998.

[7] J. Zupan. “Clustering of Large Data Sets.” *Chemometrics Research Studies Series*. Research Studies Press, 1982.

[8] A. K. Jain and R. C. Dubes. “Algorithms for Clustering Data.” Prentice Hall, 1988.

[9] B.S. Duran and P.L. Odell. “Cluster Analysis: A Survey.” *Lecture Notes in Econimics and Mathematical Systems 100,* Springer-Verlag, 1974.

[10] L. Kaufman and P. Rousseeuw. “Finding Groups in Data: An Introduction to Cluster Analysis.” Jhon Willey and Sons, 1990.

[11] Michael R. Anderberg. “Cluster Analysis for Applications.” Academic Press, New York and London, 1973.

[12] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. “Clustering Large Datasets in Arbitrary Metric Spaces.” *Technical report, University of Wisconsin-Madison*, 1998.

[13] V. Ganti, J. Gehrke, R. Ramakrishnan. “Mining Very Large Databases.” *In IEEE Computer, Volume: 32 Issue: 8*, pp. 68-75, Aug. 1999

[14] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. “Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications.” *In Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data,* Seattle,Washington, June 1998.

[15] J. Allan, A. Leouski, and R. Swan. “Interactive Cluster Visualization for Information Retrieval.” *Technical Report IR-116, Center for Intelligent Information Retrieval*, University of Massachusetts, Amherst, 1997.

[16] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Partitioningbased clustering for web document categorization. Decision Support Systems (accepted for publication), 1999.

[17] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. “BIRCH: An Efficient Data Clustering Method for Very Large Databases.” *In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pp. 103-114, Montreal, Canada, 1996.

[18] S. Guha, R. Rastogi, and K. Shim. “CURE: An efficient clustering algorithm for large databases.” *In Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 73-84, New York, 1998.

[19] G. Karypis, Eui-Hong Han, V. Kumar. “CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling.” *In IEEE Computer, Volume*: 32 Issue: 8 , pp. 68-75, Aug. 1999.

[20] R. Stepp, R. Michalski. “Conceptual clustering of structured objects: A goal-oriented approach.” *Artificial Intelligence (28)*, pp. 43-69, 1986.

[21] A. Frick, A. Ludwig and H. Mehldau. “A fast adaptive layout algorithm for undi-rected graphs.” *Proceedings of Graph Drawing 94*, LNCS 894, Springer Verlag,1995.

[22] I. Bruss, A. Frick. “Fast Interactive 3-D Graph Visualization.” *Proceedings of Graph Drawing 95*, Springer Verlag, LNCS 1027, pp. 99-110, 1996.

[23] T. R. Henry, S. E. Hudson. “Interactive Graph Layout.” *Proceedings of the ACM SIGGRAPH Symposium*, Proceedings ACM Siggraph Symposium on UI Soft-ware, 1991.

[24] Sofianto Li, Olivier de Vel, and Danny Coomans. “Comparative performance analysis of non-linear dimensionality reduction methods.”, Technical Report, James Cook University, 1995.

[25] T. Kohonen. “Self-Organizing Maps Second Extended Edition.” *Springer Series in Information Sciences, Vol. 30c* Springer, Berlin, Heidelberg, New York, 1995, 1997.

[26] M. H. Gross, F. Seibert. “Visualization of Multidimensional Data Sets using a Neural Network.” , pp. 145-159, 1993.

[27] F. Young. “Multidimensional scaling: history, theory, and applications.” Lawrence Erlbaum associates, Hillsdale, New jersey, 1987.

[28] M. Arbib, and E. G. Manes. “Arrows, Structures, and Functors: The Categorical Imperative.” Academic Press: New York, pp 93-106, 1975.

[29] S. Abramsky, Dov. M. Gabbay and T. S. E. Maibaum, Ed. “Background: mathematical structures.” *Handbook of Logic in Computer Science, Volume 1*, ISBN 0-19-853735-2, Oxford: Clarendon Press, 1992.

[30] R. Hendley, et al. “Case Study - Narcissus: Visualizing Information.” *Proceedings of the IEEE Information Visualization 95*, pp. 90-96, 1995.

APPENDIX A: ELLIPSOIDAL SURFACE PARAMETRIZATION

Parametrization of Ellipsoidal Surfaces

In this appendix we describe how to construct an ellipsoidal surface that encloses a set of N points $\boldsymbol{p}_i = (x_i, y_i, z_i)$ in space. We will compute the position, the scaling and the orientation of the ellipsoid.

Position of the Ellipsoidal Surface

The center $\boldsymbol{m} = (m_x, m_y, m_z)$ of the ellipsoid can be computed as the arithmetic average of the position of the points in space:

$$\boldsymbol{m} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{p}_i \tag{19}$$

Scaling and Orientation of the Ellipsoidal Surface

The scaling and orientation of the ellipsoid can be computed with the covariance matrix C of the set of points. The matrix C is defined as follows:

$$\begin{aligned} C &= \frac{1}{N}\sum_{i=1}^{N}(\boldsymbol{p}_i - \boldsymbol{m})(\boldsymbol{p}_i - \boldsymbol{m})^T \\ &= \frac{1}{N}\sum_{i=1}^{N}\begin{bmatrix} x_i - m_x \\ y_i - m_y \\ z_i - m_z \end{bmatrix} \cdot \begin{bmatrix} x_i - m_x & y_i - m_y & z_i - m_z \end{bmatrix} \\ &= \begin{bmatrix} \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)^2 & \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)(y_i - m_y) & \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)(z_i - m_z) \\ \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)(y_i - m_y) & \frac{1}{N}\sum_{i=1}^{N}(y_i - m_y)^2 & \frac{1}{N}\sum_{i=1}^{N}(y_i - m_y)(z_i - m_z) \\ \frac{1}{N}\sum_{i=1}^{N}(x_i - m_x)(z_i - m_z) & \frac{1}{N}\sum_{i=1}^{N}(y_i - m_y)(z_i - m_z) & \frac{1}{N}\sum_{i=1}^{N}(z_i - m_z)^2 \end{bmatrix} \end{aligned} \tag{20}$$

The scaling matrix S is then constructed using the three eigenvalues σ_1, σ_2 and σ_3 of the matrix C:

$$S = \begin{bmatrix} \sqrt{\sigma_1} & 0 & 0 \\ 0 & \sqrt{\sigma_2} & 0 \\ 0 & 0 & \sqrt{\sigma_3} \end{bmatrix} \tag{21}$$

The orientation of the ellipsoid is specified by the rotation matrix R, which is defined by the eigenvectors u_1, u_2 and u_3 of the covariance matrix C:

$$R = \begin{bmatrix} u_{1x} & u_{2x} & u_{3x} \\ u_{1y} & u_{2y} & u_{3y} \\ u_{1z} & u_{2z} & u_{3z} \end{bmatrix} \tag{22}$$

DEFINITION: A value σ is called *eigenvalue* of a $N \times N$ matrix M, if a vector $\boldsymbol{u}$ exists, such that $M \cdot \boldsymbol{u} = \sigma \cdot \boldsymbol{u}$. The vector $\boldsymbol{u}$ is defined as the *eigenvector* of the *eigenvalue* σ. A matrix of dimension N has exactly N eigenvalues and N eigenvectors.

The three eigenvalues of the covariance matrix C defined in eqn. (20) are evaluated by setting the determinant of the matrix $C - \sigma \cdot I$ to zero and solving the resulting system for σ:

$$\|C - \sigma \cdot I\| = \left\| \begin{bmatrix} C_{00} - \sigma & C_{01} & C_{02} \\ C_{10} & C_{11} - \sigma & C_{12} \\ C_{20} & C_{21} & C_{22} - \sigma \end{bmatrix} \right\| = 0 \tag{23}$$

Fig. 23 can be interpreted as a polynomial $P(\sigma)$ of degree three in the unknown σ. The three solutions σ_1, σ_2 and σ_3 of $P(\sigma) = 0$ correspond to the eigenvalues of the matrix C.

Finally, the three eigenvectors $\boldsymbol{u}_1$, $\boldsymbol{u}_2$ and $\boldsymbol{u}_3$ of C can be computed from the eigenvalues of C as:

$$\begin{bmatrix} C_{00} - \sigma_1 & C_{01} & C_{02} \\ C_{10} & C_{11} - \sigma_2 & C_{12} \\ C_{20} & C_{21} & C_{22} - \sigma_3 \end{bmatrix} \cdot \vec{u}_i = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad \text{für } i = 1\ldots3 \tag{24}$$

Since the scaling matrix only contains an average of the final scaling, it is necessary to adjust the scaling of the ellipsoid.

Adjustment of the scaling

In order to adjust the scaling matrix S it must be tested whether all the points $\boldsymbol{p}_i$ lie in the interior of the ellipsoid. If this is not the case, then it is necessary to adjust the scaling of the ellipsoid. To make the computations easier, the center of the ellipsoid must be set in the origin of the coordinate system. This is accomplished by translating and rotating all the points $\boldsymbol{p}_i$ using the position and orientation of the ellipsoid. Figure 15 illustrates this process with an example.

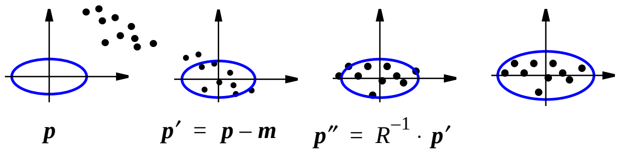


Figure 15: The points are centered on the origin

The scaling factor τ can be computed from the new coordinates (x, y, z) of the all the points $\boldsymbol{p}_i$ and the three eigenvalues σ_1, σ_2 and σ_3 of the covariance matrix C as:

$$\frac{x^2}{\sigma_1} + \frac{y^2}{\sigma_2} + \frac{z^2}{\sigma_3} = \tau \tag{25}$$

The position of a point $\boldsymbol{p}_i$ relative to the ellipsoid can be extracted from the magnitude of τ:

$\tau < 1$	The point lies in the interior of the ellipsoid
$\tau = 1$	The point lies on the surface of the ellipsoid
$\tau > 1$	The point lies outside the ellipsoid

If $\tau > 1$, then the value the eigenvalues σ_1, σ_2 and σ_3 must be increased, in order to meet the condition $\tau \le 1$.

The strategy to increase the value of the eigenvalues was constructed experimentally. The idea behind his method was to make the three semi-axis of the ellipsoid equally large, thus letting the ellipsoid approximate a sphere. If after this transformation the value of τ is still larger than one, the semi-axis are scaled by a factor of τ, so that the point is guaranteed to lie on the surface of the ellipsoid.

This strategy can be implemented in four steps:

- Increase the smallest semi-axis σ_{min} up to the largest semi-axis σ_{max} and check whether the condition $\tau \le 1$ can be met.
- Increase the medium semi-axis σ_{mid} up to the largest semi-axis σ_{max} and check whether the condition $\tau \le 1$ can be met.
- Increase both the smallest semi-axis σ_{min} and the medium semi-axis σ_{mid} up to the largest semi-axis σ_{max} and check whether the condition $\tau \le 1$ can be met.
- If none of the previous steps was successful, re-scale the three semi-axis σ_{min}, σ_{mid}, and σ_{max} by a factor of τ.