Proc. 2nd Symposium on Large Spatial Databases (SSD '91)

The Performance of Object Decomposition Techniques for Spatial Query Processing

Hans-Peter Kriegel, Holger Horn and Michael Schiwietz

Praktische Informatik, Universität Bremen, D-2800 Bremen 33

Abstract:
The management of spatial data in applications such as graphics and image processing, geography as well as computer aided design (CAD) imposes stringent new requirements on spatial database systems, in particular on efficient query processing of complex spatial objects. In this paper, we propose a two-level, multi-representation query processing technique which consists of a filter and a refinement level. The efficiency of spatial query processing is improved considerably using the following two design paradigms: first, divide and conquer, i.e. decomposition of complex spatial objects into more simple spatial components such as convex polygons, triangles or trapezoids, and second, application of efficient and robust spatial access methods for simple spatial objects. The most powerful ingredient in our approach is the object decomposition. Applied to the refinement level of the query processor, it substitutes complex computational geometry algorithms by simple and fast algorithms for simple components. In this paper, we present four different decomposition techniques for polygonal objects. The second part of the paper consists of an empirical performance comparison using real and synthetic data. The four types of decomposition techniques are compared to each other and with the traditional approach with respect to the performance of spatial query processing. This comparison points out that our approach using object decomposition is superior to traditional query processing strategies.

1 Introduction

The demand for using database systems in application areas such as graphics and image processing, computer aided design (CAD) as well as geography and cartography is considerably increasing. The important characteristic of these applications is the occurance of spatial objects. The management of spatial objects imposes stringent new requirements on spatial database systems. One of the most challanging requirements is efficient query processing of complex spatial objects.

The typical object type that occurs in the above mentioned applications are two- or three-dimensional spatial objects. Points, lines, or rectangles are known as simple spatial objects, because their complete description is given by only a small number of parameters. Semantically complex objects with an application specific complexity, such as contour lines, limits of lots, and contours of CAD objects have the shape of simple polygons. Complexity properties of such polygonal objects, such as the shape, the number of vertices, or the smoothness of the contour are difficult to predict. Additionally, as a general property of polygons, holes have to be taken into account for a general handling of objects occurring in geographic information systems, e.g. to model areas of land containing lake areas. In order to support the above type of spatial applications, the ability to manage simple polygons is fundamental to a spatial database system. In this paper, we would like to present and to evaluate a query processor based on spatial access methods (see for example [NHS 84], [See 90]) and computational geometry techniques [PS 85].

In the next chapter we introduce a query processing mechanism using a filter technique based on spatial access methods. The basic ingredient for achieving performance improvements in this query processing mechanism is the introduction of redundancy [Ore 89]. This is the subject of chapter 3. A special type of introducing redundancy in object representation is the socalled *structural decomposition.* Chapter 4 describes four different structural decomposition techniques for SPHs. In chapter 5 we describe the processing of spatial queries based on object decomposition in more algorithmic detail. Chapter 6 contains a comparison of the different structural decomposition techniques with respect to their performance within spatial query processing. The paper concludes with a summary that points out the main contributions and gives an outlook to future activities.

2 Query processing using filter techniques

In this chapter, we will introduce a special type of polygonal objects and a query processing mechanism for managing large sets of such objects.

2.1 Spatial objects and spatial queries

The types of spatial objects we consider is the class of *simple polygons with holes* (SPH, see figure 1). A polygon is called simple if there is no pair of nonconsecutive edges sharing a point. An SPH is a simple polygon where simple polygonal holes may be cut out from the enclosure polygon. From our experience, the class of SPH is adequate for GIS applications (see [Bur 87]) and most 2D CAD/CAM applications.

Figure 1: simple polygon simple polygon with holes

Queries in spatial applications generally refer to spatial and nonspatial data. Spatial data can be classified into geometrical and topological aspects whereas nonspatial data is given by alphanumerical data related to spatial entities. Geometric data describes properties such as the spatial location, size, and shape of spatial objects. Topological data describes properties such as connectivity, adjacency and inclusion modelling relationships between geometric data. By the way, it is not necessary to store topological data explicitely, because it can be derived from geometric data by formulating suitable query conditions. Most spatial query conditions describe such topological aspects between stored objects and the query object. Additionally, spatial queries not only retrieve data, but also may construct new objects. Usually these objects are displayed and not necessarily stored in the database.

From the literature no standard set of geometric queries fulfilling all requirements of spatial applications is known [SV 89]. Thus it is necessary to provide a small set of basic spatial queries which are efficiently supported by the database system. Application specific queries, e.g. in [Oos 90], typically using more complex query conditions, can be decomposed into a sequence of such basic spatial queries. We present the following set of basic spatial queries:

- **PointQuery:** Given a point $p \in E^2$, find all SPHs in the database where $p \in$ SPH.
- **WindowQuery:** Given a rectilinear window $w \subseteq E^2$, find all SPHs in the database where $w \cap \text{SPH} \neq \emptyset$.
- **RegionQuery:** Given an SPH* $\subseteq E^2$, find all SPHs in the database where SPH* $\cap$ SPH $\neq \emptyset$.
- **EnclosureQuery:** Given an SPH* $\subseteq E^2$, find all SPHs in the database where SPH* $\supseteq$ SPH.
- **ContainmentQuery:** Given an SPH* $\subseteq E^2$, find all SPHs in the database where SPH* $\subseteq$ SPH.
- **IntersectionQuery:** Given an SPH* $\subseteq E^2$, compute the intersection of SPH* with all SPHs in the database.

As an example consider the following query: Given an area bounded by two latitudes and two meridians. Find the most populated city within this area and the state, this city belongs to. This query can be evaluated by initially enforcing a window query yielding the set of all cities lying within the specified area. After computationally determining the most populated one a point/enclosure query finds out the unique state to which this city belongs.

2.2 Query processing supported by access methods and computational geometry

A typical property of spatial queries is their restriction to a specific spatial location in data space. Only that location and some limited neighbouring area is essential for the evaluation of most spatial queries. The window query is a typical example for such a query (see figure 2).

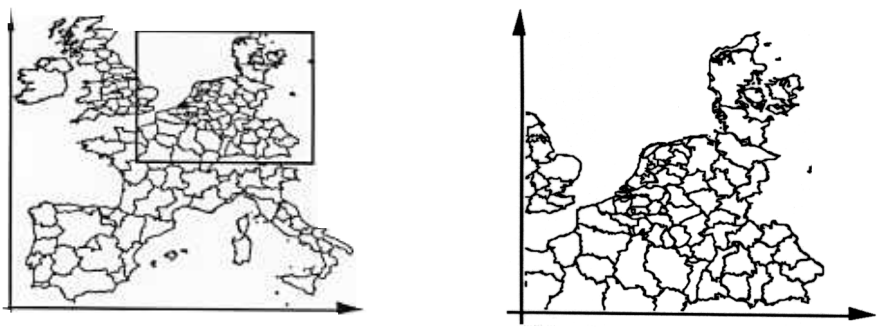

Figure 2: Zooming into a query window

Obviously, objects (SPHs) lying close together in dataspace are often accessed jointly by a window query. The same holds for the other basic queries defined above. Therefore a physical clustering of spatial objects with respect to their spatial location is essential for providing efficient locality based query processing. This type of spatial clustering is supported by spatial access methods (SAM), introduced below. In the absence of such a spatial clustering no spatial locality can be exploited by a query processing algorithm. Every single stored object has to be evaluated against the query condition leading to poor performance that is further decreasing with an inceasing number of stored objects and an increasing object complexity. Therefore, one essential ingredient of an efficient query processing within a spatial database system is spatial clustering of the objects.

In the past few years many spatial access methods were developed which provide the organization of large sets of simple spatial objects on secondary storage. The most simple class of spatial objects managed by such access methods are (multidimensional) point objects. The grid file [NHS 84], PLOP-hashing [KS 88], the BANG file [Fre 87], and the buddy tree [SK 90] are well known representatives of this class of access methods. A survey can be found in [SK 90]. There are three basic techniques for extending multidimensional point access methods (PAMs) to multidimensional spatial access methods (SAMs) for rectangles [SK 88]: clipping, overlapping regions and transformation.

Another important characteristic of a SAM is the type of spatial objects it is able to handle directly, i.e. the type of objects which are exactly represented. All spatial access methods proposed up to now, are restricted to the storage of simply shaped objects such as cells of a fixed grid (grid cells for short) [OM 86], rectilinear rectangles (with their sides parallel to the axis, rectangles for short) [Gut 84], [NHS 84], [SK 88], [SK 90], [BKSS 90], spheres [Oos 90] or convex cells (convex polygons) [Gue 89].

However, no spatial access methods is available for more complex spatial objects and particularly is not for the class of SPH. In order to provide an efficient access method for complex spatial objects, a 'brute force' approach was applied up to now. Any spatial object is placed within a rectilinear rectangle or convex polygon of minimum shape forming a *container* for that object, yielding a socalled conservative approximation. A simple spatial object is called a container iff any point inside the contour of the complex spatial object is also contained in the container object. Those containers are selected according to their suitability to be handled by one of the spatial access methods mentioned above.

As simple containers just provide conservative approximations, query processing on complex spatial objects has to procede in a two-step manner. The first step, the socalled *filter step*, reduces the entire set of objects to a subset of candidates using their spatial location. The filter step is based on spatial access methods managing container objects using the following property: if the container does not fulfill the query condition, so does not the object itself. However, because container objects provide no exact representation, this filter step does not exactly evaluate the query, but only yields a set of candidates, which may fulfill the query condition. Therefore, these candidates have to be examined in a second step, called *refinement step*. This step applies complex algorithms known from the field of computational geometry to the original spatial objects and detects exactly those objects finally fulfilling the query condition.

Consider the following Point Query as an example for this kind of query processing. Two objects remain to be examined in the refinement step, but only one of them fulfills the query condition (figure 3).

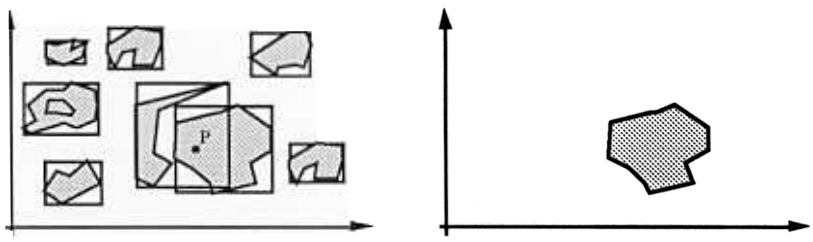

Figure 3: Example for the Point Query

At first glance, this brute force approach of coupling a spatial access method and computational geometry algorithms, seems to be a good method of query processing. However, more detailed considerations reveal the main disadvantages of this approach: in the case of a bad approximation of an object by its container, a large number of 'false drop' candidates, i.e. objects not fulfilling the query condition, have to be refined. On the other hand, the refinement of one single object is very costly particularly if the object complexity is high, because complex and time-consuming computational geometry algorithms have to be applied. We have to consider these two aspects when tuning the performance of spatial query processing.

In the next section we will examine both steps of query processing in more detail to show possible ways of performance improvements.

2.3 Performance improvements of query processing

The filter step

The performance of the filter step considerably depends on the quality of the spatial object approximation by the container used for filtering issues. The approximation quality is defined as the amount of area covered by the container but not by the object itself. As containers, e.g. minimal bounding boxes, are simple spatial objects, they cannot exactly represent complex spatial objects without introducing additionally covered area. Minimizing the amount of that area will directly (proportionally) improve the filter step. Also more complex containers may yield a better approximation. The only requirement to the container type is the ability to be efficiently managed by a SAM. Particularly for rectangles there are very efficient access methods, e.g. the R*-tree [BKSS 90]. Therefore, we propose rectilinear polygons as a particular type of container. They can be formed by a set of rectilinear boxes, at the expense of introducing redundancy in object representation. Using a number of containers to represent one object is the basic idea of redundant object representation. For SAMs some limited amount of redundancy leads to better query performance, as we will see in chapter 6.

The refinement step

The performance of the refinement step depends on the number of refined objects as well as on their complexity. Minimizing the number of objects to be refined is the task of the filter step. Therefore, object complexity is the issue to be examined here. The more complex the spatial objects are, the more time consuming are the computational geometry algorithms needed for query evaluation. A simplification of the refined objects with respect to their complexity may lead to a better overall performance of the refinement step even if a limited amount of redundancy has to be handled.

Summarizing, the main goals in performance improvement of filter based query processing are the following:

1. Improvement of the accuracy of the filter step to minimize the number of candidates.
2. Improvement of the refinement step by using objects considerably simpler than the original SPHs to speed up computational geometry algorithms.

The above considerations show, that for improving the performance of spatial query processing, we have to give up using one single container for every complex spatial object. Therefore, our objective is to represent the SPHs within the spatial access methods using a number of containers inducing redundancy in object representation.

A more detailed consideration of different types of redundant representations of SPHs is subject of the next chapter.

3. Object representation based on redundancy

The basic idea of any redundancy based object representation is to improve query performance by shifting time requirements from query processing to update and restructuring operations. Retrieval operations typically occur considerably more often than update operations, e.g. insertions. Thus it is worth to invest more time in preprocessing which is saved in a manyfold way in query processing. Specifically, we will generate a new type of object representation in preprocessing which is time saving in query processing. In more detail, the preprocessing step calculates a redundant object representation given by a decomposition into less complex components. This leads to a better filter approximation and more efficient computational geometry algorithms within the refinement step.

In the following, we will present different types of redundant object representations for the class of SPHs known from the literature [Ore 89].

Minimum bounding boxes (no redundancy):
The object management using one object container in the filter step, coupled with a refinement step on the original complex object representation is a widely used traditional method (see [Gut 84]). This approach is called 'identity' representation from now on. Without introducing redundancy this is the only approach managing complex spatial objects by SAMs preserving spatial location and exploiting spatial clustering. The disadvantages of this approach, leading to a bad query performance, have been outlined before.

Redundancy induced by cells of a fixed grid:
In the quad tree / z-value ([Sam 84], [Ore 89] approach a container for a spatial object is formed by exactly those cells of a fixed grid which have a common intersection with the object. As a cell of a fixed grid can be efficiently represented by a bitstring, i.e. a z-value, representing the recursive grid partitioning of the dataspace, those cells can be efficiently stored using an one-dimensional access method, e.g. a B-tree. Inducing redundancy is intended by forming a smaller and more complex container and therefore increasing the performance of the filter step. As the object complexity generally is not decreased using this approach, the refinement step is not improved.

Redundancy induced by grid and object structure:
A further approach is taken by grid based methods which are not restricted to a predefined grid resolution, but taking account of the object location and structure. Similiar to the edge-quadtree [Sam 84], an object is partitioned into grid cells, socalled base grid cells. A base cell is the largest cell formed by recursive grid partitioning containing parts of the object which fulfill a predefinable complexity condition. Therefore, no minimum cell resolution can be guaranteed. The resolution depends on the shape of the object. Thus the partitioning process can typically produce cells of very small area. The refinement step, however, contrary to original grid cell methods using a fixed grid resolution, is tuned by the occurrence of very simple objects represented by each grid cell. The amount of redundancy, however, in this method essentially depends on the object structure, e.g. the distance of two vertices, and can arbitrarily grow.

Redundancy induced by structural decomposition:
The usage of grid cells representing spatial objects has its origin in the lack of suitable spatial access methods for higher dimensional dataspace. Grid cells represented by bit strings form a transformation to a one-dimensional linearly ordered dataspace. However, grid cells are bound to a fixed partitioning scheme and therefore provide no location independent object approximation as minimum bounding boxes do. Therefore the most promising approach introducing redundancy to object representation methods is a *structural decomposition* of complex spatial objects into simpler components. The term 'structural' expresses that the decomposition is oriented on the boundary of the polygonal object. Similar to the original bounding box approach, the components are managed by a SAM by placing them into containers, e.g. minimum bounding boxes. Structural decompositions provide a high degree of choices for component types and decomposition algorithms. Typical object types for components are convex polygons, trapezoids, triangles and rectangles. Choosing a proper decomposition algorithm will improve both, the filter step and the refinement step. The filter step performance will benefit from a better overall object approximation by a container approximation of each single component. However, the problem of multiple representation of one and the same object induced by redundancy has to be regarded. Additionally, the refinement step will be improved by simpler objects, which can be processed faster by computational geometry algorithms.

In the next chapter, we present four different decomposition techniques for SPHs. The different redundant representations for an SPH produced by these decomposition algorithms will be compared within a spatial query processor with respect to their query performance in chapter 6.

4. Structural decomposition techniques

The goal of the following sections is to examine different structural decomposition techniques with respect to their performance for spatial query processing. At first, a catalogue of important properties is introduced which allows to distinguish and classify the different types of decomposition algorithms for polygonal objects. Then we will describe four different algorithms in more detail.

4.1 Properties of structural decomposition techniques

The basic properties of a structural decomposition method can be divided into those describing qualitative and those describing quantitative aspects of the method. Qualitative aspects are the type of generated decomposition components and the distinction between partitioning or covering of spatial objects. A decomposition is called a partition iff all components are pairwise disjoint. Otherwise, it is called a covering. The application of specific techniques from the field of computational geometry (plane sweep, divide and conquer, etc.) is another qualitative aspect of a decomposition technique.

Quantative aspects are the number of generated components with respect to the complexity of the object and the quality of container approximation of these components. The time needed to decompose one object and to apply computational geometry algorithms to the decomposition components is a further quantitative aspect of a decomposition technique.

Abstracting from the specific properties mentioned above, decomposition techniques can be classified with respect to the complexity and the number of generated decomposition components. However, decomposing into very simple components will lead to a high number of components and vice versa.

The basic idea of comparing different decomposition techniques (see chapter 6) is to experimentally evaluate which grade of object decomposition leads to best performance in spatial query processing. What is an adequate type of object components with respect to complexity of components and the amount of redundancy?

Therefore, we selected and implemented four different decomposition algorithms which will be explained in detail in the next chapter.

Most decomposition techniques known from the field of computational geometry have been developed under requirements different from the application of efficient (spatial) query processing. In order to achieve best query performance, it is necessary to take into account a *number* of requirements of object representation. The algorithms proposed in computational geometry ([CD 85], [KeSa 85]) are mostly optimal with respect to one of those properties, e.g. minimal number of components, but totally ignore other aspects, e.g. the run time. Therefore, it was necessary to develop particular decomposition algorithms for query processing of complex spatial objects. The most important requirements are:

Low number of components
Increasing redundancy induced by a high number of components affects the performance of the filter step which has to manage a significantly larger number of containers. Therefore, another important goal is to minimize the number of generated components.

Good run time performance
Whenever a new object is inserted into the database, a decomposition of that object must be perfromed. Therefore, algorithms optimal in the number of components which requires an exponential run time, e.g. proposed in [PS 85] or other papers, are quite unacceptable. Thus decomposition algorithms with a run time of low order have to be provided.

Good container approximation
As outlined in chapter 2, the evaluation of a spatial query consists of the filter and the refinement step. The filter step, based on a object container representation, yields the set of candidates to be examined in the refinement step. Therefore, it is of crucial importance for the filter step to minimize the 'dead space' between a spatial object and its container and thus to reduce the number of components passing the filter step. For achieving a good container approximation, object decomposition techniques must supply components that can be well approximated by containers, e.g. minimum bounding boxes.

Small amount of storage
Until now, we only focussed on efficient processing of spatial queries. Nevertheless, limiting the amount of additional storage required for the redundant object representation is important as well.

Ease of implementation
The integration of object decomposition techniques into a real spatial database system demands in the developement of robust algorithms, easy to implement and maintain.

Obviously, there are many different criteria influencing the quality of a decomposition method. Without further examination the importance of any of these criteria is not foreseeable. Therfore, we performed an empirical comparison (chapter 6) to evaluate which method achieves best query performance.

In the next sections, we will introduce four selected decomposition methods. We will give a brief algorithmic description and try to depict particular properties with respect to the criteria outlined above. One important premise for the developement of the algorithms was to use minimum bounding boxes as containers for the components.

4.2 Decomposition into convex polygons

As geometric algorithms for the type of convex polygons are more efficient than those for arbitrary SPHs, we consider as a first approach the decomposition of an SPH into a set of convex polygons. The basic idea is to

transform the original SPH into a simpler but equivalent geometric object and then decompose this new object with an appropriate algorithm. This idea leads to the following two step algorithm [DHH 90]:

Step 1: Transform the original SPH P into a polygon P´ describing the same infinite set of points as P but containing no holes. This polygon P´ is simple with the exception that edges may overlap. From now on, those polygons are called simple*.

Step 2: Decompose the simple* polygon P´ into a set of convex polygons.

Within the following algorithm the two techniques, *transformation* (step 1) and *divide and conquer* (step 2), from the field of computational geometry are applied (see [PS 85]).

Step 1: Hole integration

The objective of the first step is to integrate all holes of the given SPH *P* into the enclosure polygon of *P* without changing the infinite set of points described by the object. This will be achieved by sending out *hole integration rays* from the holes of the SPH towards the enclosure polygon until all holes are removed from the original SPH. For a successfull termination of the hole integration step it is necessary to define an integration order on the holes. This will be achieved by building up the *convex hull* over all the holes of the polygon and sending out integration rays from the holes having points on this hull towards the enclosure polygon. Thus, a subset of the holes are integrated into the enclosure polygon and the algorithm continues with building up the convex hull of the remaining holes and so on until no more holes are left. At the end of step 1 a simple* polygon is produced describing the same (infinite) set of points as the original SPH.

An important feature of the algorithm is that the integration rays, i.e. all new segments, should be parallel to one axis of the coordinate system whenever possible. This is due to the fact that all decomposition components generated by step 2 of the algorithm will be approximated by rectilinear rectangles. Figure 4 gives an example of the effect of step 1.

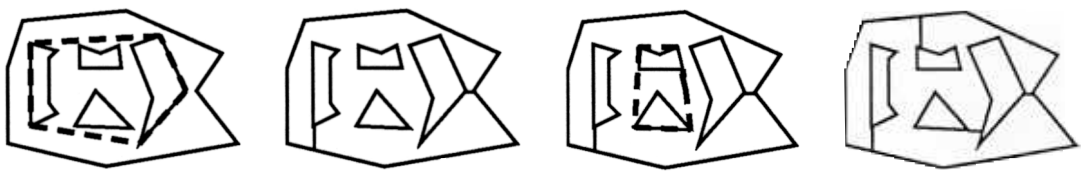

Figure 4: Example for the hole integration step

Step 2: Decomposition of the simple* Polygon

After removing the holes using step 1, the task of step 2 is to perform the actual decomposition of the object *P´* into convex componets. As mentioned before, the only difference between simple polygons and simple* polygons is the existence of overlapping edges. Therefore, most algorithms for the decomposition of simple polygons into convex parts (see [CD 85]) can also be applied to simple* polygons with only slight modifications. How can we decompose a simple polygon into convex parts? The vertices of a given simple polygon which display a reflex angle, called *notches*, will play the crucial role in the following, because the result objects (convex polygons) must not contain such vertices. So the basic idea of step 2 is to remove each notch by means of simple line segments drawn from the notch. As in step 1, line segments should be parallel to one axis whenever possible. The algorithm uses the technique of *divide and conquer* because after removing one notch from the original polygon it will be applied to both produced result components in the same way until no more notches remain and a set of convex components is achieved. Applying step 1 and 2 to the original SPH gives the result set of convex components. Figure 5 shows the effect of step 2.

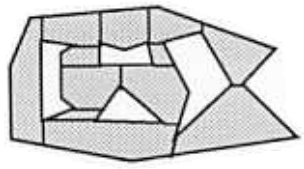

Figure 5: Example for the convex decomposition of the simple* polygon

We can summarize the properties of the presented decomposition algorithm: it is easy to implement, has a good run time performance ($O(n \log^2 n)$, where n is the number of vertices) and the number of produced components in the worst case is twice the optimum number of convex components (see [DHH 90]).

4.3 Decomposition into trapezoids

The second partitioning method we take into consideration is the decomposition of an SPH into a set of trapezoids introduced by Asano/Asano [AA 83]. The components produced by this algorithm are formed as trapezoids containing two horizontal sides. This property provides a good container approximation by means of rectilinear rectangles. The algorithm uses the plane sweep technique known from the field of computational geometry. The basic idea is, to send out for each vertex one or two horizontal rays into the interior of the polygon to the first edge encountered. In the following we give a brief description.

As mentioned before the algorithm uses the plane sweep technique, i.e. the vertices will be passed and handled with increasing y-coordinates, for example. Thus, the entire algorithm consists of two steps:

Step 1: Sorting the vertices of the given SPH (enclosure polygon and holes) builds up the *event point list* , i.e. a sorted list of all the points which have to be treated by the algorithm.

Step 2: Processing the event point list by switching from one event point (vertex) to the next sending out partition rays (*event point scheduling*). Within this process, each ray and its successor form one trapezoid[1]. Thus, the set of component objects is gradually generated.

Applying step 1 and step 2 to the original SPH produces the resulting set of trapezoids. Figure 6 shows an example of the effect of the algorithm.

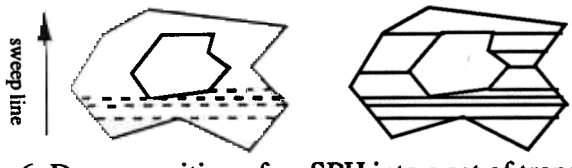

Figure 6: Decomposition of an SPH into a set of trapezoids

Summarizing, the main properties of the presented decomposition technique are: It is very easy to implement and has a good run time performance of $O(n \log n)$ (where n is the number of vertices). The number of produced components is three times the optimum number of components in the worst case, but it is quite better for most SPHs [AA 83].

4.4 Triangulation

The basic concept of decomposition of arbitrary SPHs is generating a low number of simple object components which support fast query processing. Triangles are very simple objects with a fixed length description and are easy to handle with computational geometry algorithms. The triangulation algorithm introduced here guarantees the generation of a minimum number of triangles for a given SPH where the triangles introduce no new vertices. The algorithm works in a three step manner:

[1] In some cases degenerated trapezoids, i.e. triangles with one horizontal side may be produced.

Step 1:
Triangulate the SPH using the Delaunay triangulation [Del 34]. The Delaunay triangulation works on the set of vertices of the SPH and generates a triangulation of its convex hull, see figure 7a/b. It fulfills the socalled Lawson criterion which guarantees homogenous triangles in the sense that the deviation of the angles of a triangle is small. Thus the degeneration of the shape of the triangles is restricted. The resulting component triangles may be completely inside the SPH or may be completely outside of the SPH or may have intersection points with edges of the SPH. Figure 7a/7b depicts that the Delaunay triangulation does not necessary represent the original SPH, i.e. this only happens if the SPH is convex and has no holes.

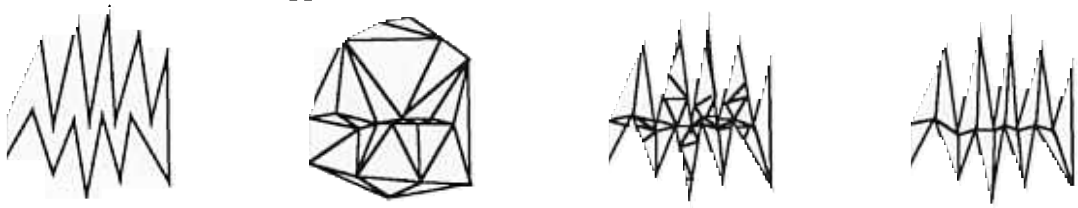

Figure 7: a) Sample SPH (19 vertices) b) Delaunay Triangulation (27 Triangles) c) Provisional Triangulation (50 Triangles) d) Final Triangulation (17 Triangles)

Step 2:
In the second step a plane sweep algorithm [KHS 90] is applied which detects all intersections of the set of triangles with the edges of the SPH. The intersections are removed by decomposing a triangle with intersection points into a set of subtriangles where no more intersections occur, see figure 7c. The number of the additionally generated triangles depends on the number of intersections. At this point, the algorithm does not restrict to the vertices of the SPH but has introduced new vertices, socalled 'Steiner points'.

Step 3: In the third step all Steiner points are completely removed by examining each set of Steiner points lying on an edge of the SPH and merging all triangles belonging to these Steiner points into a single triangle. Figure 7d depicts the final solution restricting to the vertices of the original SPH.

The properties of the triangulation introduced above can be summarized as follows: A minimum number of triangles is guaranteed from the algorithm. There are no new vertices generated. The run time performance is $O((n+k)\log(n+k))$, see [KHS 90], where n is the number of vertices and k is the number of intersections of the triangles from step 1 with the edges of the SPH. Generally, there is no guarantee for triangles to contain horizontal or vertical edges, which are essential for the quality of a container approximation.

4.5 Heterogeneous decomposition

The idea of this decomposition technique is to represent an SPH by components even more simple than arbitrary triangles or trapezoids. With respect to a container approximation by its bounding box, the most simple type of component is a (rectilinear) rectangle. As this type of components is insufficient for the representation of arbitrary SPHs, i.e. an arbitrary SPH can not be exactly represented by a set of rectangles, it is necessary to use further types of components. Therefore, the decomposition technique is called *heterogeneous*. With respect to a good approximation of the components by bounding boxes, we choose rectilinear rectangles and rectilinear triangles, i.e. triangles with two edges parallel to the axes, as decomposition components. To provide a unique representation of the area covered by an arbitrary SPH, an additional type of components managing notches, i.e. a particular shape property of polygonal objects described before, is necessary. This type of component is called a peak. Particularly for real applications this type of component turns out to occur only in exceptional cases. Therefore, a more detailed description of peaks is ignored in this paper. Figure 8 shows an example for a heterogeneous decomposition of a polygonal object.

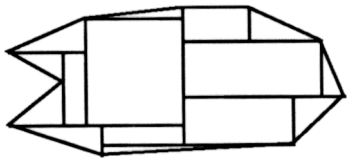

Figure 8: Decomposition of an object into a set of rectilinear triangles and rectangles

The complete algorithm for decomposing an SPH into the three types of components mentioned above consists of 3 steps:

step 1: Remove the peaks representing them by means of peak components. This step builds up a new SPH without peaks.

step 2: Remove all non rectilinear edges of the SPH by using rectilinear triangles. Thus, a set of rectilinear polygons is generated.

step 3: Decompose all rectilinear polygons into rectilinear rectangles.

Figure 9 depicts snapshots of the result of the algorithm (step 1-3) for a sample polygon. Additionally let us mention that the decomposition process can as well be performed by a plane sweep type algorithm. However, this algorithm is considerably more complicated, and therefore it is not presented here.

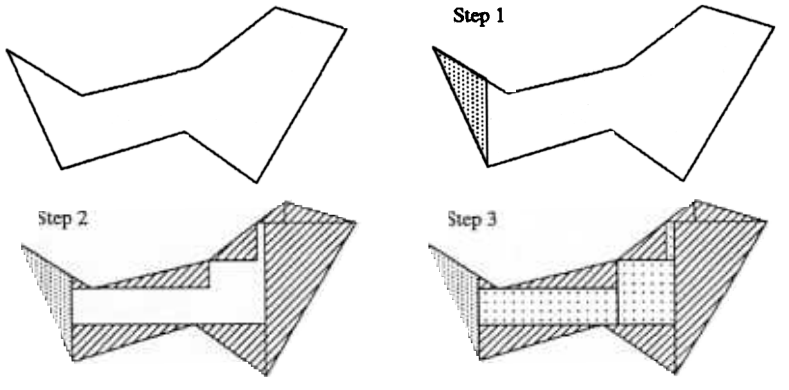

Figure 9: Heterogeneous decomposition in a 3 step algorithm

One important property of this heterogeneous decomposition technique is that the generated components are very simple. Therefore, we expect that they can be processed very fast by geometric algorithms. However, this is enforced by a larger number of components than in other decomposition methods (twice the number of components in practice). As the algorithmic complexity strongly depends on the structure and on the shape of the decomposed object, an exact run time investigation of the above algorithm cannot be performed analytically.

5. Query processing based on object decomposition

The objective of this chapter is to describe in more algorithmic detail the processing of different spatial queries based on a decomposed object representation. In particular we will take a closer look at the insert operation, the point query and the window query. These operations will be analyzed with respect to their performance in chapter 6. The algorithms described below are not restricted to a specific decomposition technique, but hold for any redundant spatial object representation.

5.1 Insert operation

The task of the insert operation is to add a new spatial object, i.e. an SPH, to an existing set of spatial objects, managed by a given access method. This operation consists of three steps.

Within the first step, the new object is decomposed into a set of simpler components (e.g. convex polygons, triangles) by applying one of the decomposition algorithms introduced in the previous section. Then, in step two, the minimum bounding box, i.e. the container object, for each of those components is generated. Furthermore, a unique identifier (a surrogate) for the original SPH is determined and assigned to all of its components. This identifier represents the correspondence between the original spatial object and its decomposition components. Thus, the records describing the components consist of three parts: the minimum bounding box of the component, the representation of the decomposition component, and the object identifier. In a last step, all these records belonging to the original SPH are inserted into the database using the insert algorithm of the SAM.

The result of the insertion operation is a new data file which now contains the components of the inserted SPH. The components refering to the original SPH are labeled by their unique identifier.

As expected, it turns out that the insertion cost using object decomposition is higher than using no decomposition. However, the decomposed representation of spatial objects may improve spatial query processing by an order of magnitude as we will see in chapter 6.

5.2 Point query

The result of a point query consists of all stored spatial objects containing a given query point. As described in chapter 1 the processing of any spatial query consists of two steps: The filter step and the refinement step.
The filter step of a point query asked on a decomposed object representation using a SAM yields all those components whose bounding box contains the query point. They are supplied by evaluating a point query against the SAM. The refinement step sequentially examines these candidates performing a computational geometry algorithm on the exact component representation ('point-in-object' test). If this test yields 'true', the identifier of that component record is added to the result. After examination of all candidate records the query is finished.

5.3 Window query

The window query yields all the spatial objects intersecting a given query window. Similarily to the point query the filter step of a window query is based on the SAM: A window query is performed on the file of components yielding all those components whose bounding box intersects the query window. Within the refinement step the exact representation of all these candidates is tested against the query window, i.e. a computational geometry algorithm is performed on the exact representation of each component. If there is an intersection, the object identifier of a component record is added to the result. Contrary to the point query the described algorithm of query processing has to deal with object redundancy. In general, there may be a number of different components labeled with the same identifier and therefore refering to the same spatial object intersecting the query window (see figure 10).

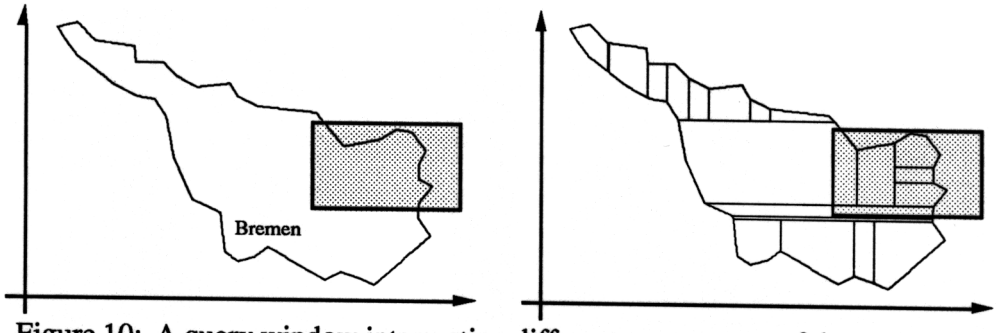

Figure 10: A query window intersecting different components of the same SPH

In such a case, the same identifier, i.e. the same spatial object, is handled more than once by the time consuming refinement algorithm. To avoid a multiple refinement of the same object and a duplicated output of

objects we propose the following strategy. We use a one dimensional main memory search structure, e.g. an AVL-tree or a one-dimensional hashing scheme, to manage the resulting object identifiers. Whenever the identifier of an object is added to the result, it is also inserted into this search structure. Then, before applying the refinement step to a filtered object, the corresponding object identifier is retrieved in the search structure. If it occurs in the search structure, an unnecessary refinement step is prevented. By means of this technique we avoid duplicates in the result and do not perform redundant refinement operations.

6. Performance comparison of decomposition techniques

After presenting different structural decomposition techniques as well as query processing algorithms using these techniques, it is the main goal of this chapter to compare the different techniques to each other and to the original, undecomposed representation. Thus, we can analyse whether it is worth it to decompose complex spatial objects and which decomposition technique is best suitable. We will present a series of tests which we ran with Modula-2 implementations on SUN 3/60 workstations under UNIX. The main question we want to answer is whether decomposition techniques lead to better query performance and which technique achieves best performance improvements in spatial query processing. The comparison consists of two parts: Part one (section 6.2) compares the different decompositions among themselves with respect to their number of produced components, their quality of container approximation and their amount of storage. Then in part two of the test (section 6.3) the different decomposition techniques are investigated with respect to query processing time.

6.1 Selection of test data

One basic problem of all empirical performance comparisons is the selection of an appropriate standardized set of test data. The best choice is to examine data files used daily in real applications. Thus, we tried to provide large sets of spatial data, e.g. digitized maps used in existing geographic information systems. Additionally, large sets of synthetic data, i.e. polygons, were generated by a tool in order to facilitate testing the query processing system under a wide range of varying data. To describe the test files used for the comparison, it is necessary to provide a set of parameters which characterizes single polygons and sets of polygons. For single polygons we choose the number of vertices, the number of holes, the size and the shape complexity as parameters. The shape complexity is characterized by the class the polygon is a member of, i.e. convex, starshaped, simple polygons and simple polygons with holes (SPH) (see [PS 85]). For sets of polygons, the number of polygons, their cover, and their distribution in data space are additional paramters to be investigated. The cover of a set of objects is the sum of the size of the objects relative to the size of the data space. In particular, we selected the following files for our comparison.

Name	Num. of Objects	Cover	Distrib.	Num. of Vertices	Num. of Holes	Shape
europe	471	0.27	real	95.1	0.02	SPH
sph_85_1	1000	1.00	uniform	85.0	2.00	SPH
sph_85_10	1000	10.00	uniform	85.0	2.00	SPH
star_20_1	1000	1.00	uniform	20.0	0.00	starshaped
star_20_10	1000	10.00	uniform	20.0	0.00	starshaped

Table 1: The test files

The first file ('europe') consists of 471 polygons representing the counties of the European Community, see table 1[2]. In addition to the examination of this real data file we generated two pairs of test files by a tool, each

[2] We thankfully acknowledge receiving this data file from the 'Statistical Office of the European Communities'

of them consisting of 1000 polygons. The 'sph_85' files contain SPHs, whereas the 'star_20' files contain simple starshaped polygons without holes. We selected different covers for the files, cover 1 and 10, to simulate different degrees of object overlap, e.g. occurring in multi attribute maps. The objects of the four synthetic files are distributed uniformly in dataspace due to the fact that the performance of the R*-tree which is used as a SAM for the bounding boxes is independent of the data distribution [BKSS 90].

6.2 Comparison of decomposition techniques with respect to structural properties

Within the first part of the comparison we examined the different decomposition techniques of chapter 4 with respect to the number of generated components, the quality of container approximation and the amount of storage. Table 2 contains the test results for the files presented in section 6.1.

europe	ident	convex	trapezoids	triangles	heterogen
Num. of Components	471	22296	44332	44218	81338
Approximation	2.15	1.31	1.20	3.13	1.25
Amount of storage	1.00	2.15	2.77	2.63	3.73

sph_85_1/10	ident	convex	trapezoids	triangles	heterogen
Num. of Components	1000	51149	83998	85004	198305
Approximation	1.72	1.31	1.10	3.13	1.15
Amount of storage	1.00	2.99	3.00	3.03	5.19

star_20_1/10	ident	convex	trapezoids	triangles	heterogen
Num. of Components	1000	7764	18958	17958	34429
Approximation	1.41	1.21	1.11	3.06	1.19
Amount of storage	1.00	2.06	2.79	2.64	3.67

Table 2: Test results for the structural properties of the decomposition techniques

For the interpretation of the results presented in table 2, we start with the analysis of the real data file 'europe'.

Considering the degree of redundancy, i.e. the number of components introduced by decomposition, shows that the number of generated components essentially depends on the type of objects stored in the file. For the trapezoid and the triangle decomposition the number of components is approximately the same as the number of vertices in the original object (see section 4.3 and 4.4). The convex decomposition generates about half, the heterogeneous decomposition about twice the number of components compared to the trapezoid and triangle decomposition. The number of convex components essentially depends on the shape, i.e. the number of notches, of the SPHs, whereas the heterogeneous decomposition generates one triangle and one rectangle for each vertex of the SPH in most cases.

Considering the quality of container approximation, the bad value of 2.15 for the identity (undecomposed) representation is conspicuous. Within query processing, such a bad value leads to frequent application of the refinement step, which is particularly time consuming due to the undecomposed representation. The application of decomposition techniques causes much better approximation values within the range of 1.2 - 1.3 for the convex, the trapezoids and the heterogeneous decomposition. This is caused by the application of partition rays parallel to one axis of the coordinate system. Using the triangle decomposition leads to values larger than 3, because rectangles are no proper type of container for triangles.

The influence of the number of components and the quality of container approximation on the performance of spatial query procesing is evaluated in the next section.

The reason for the introduction of redundancy introduced by structural decomposition techniques was to speed up spatial query processing. However, this type of object representation leads to a higher amount of

(secondary) storage than the identity representation. For real data, the convex representation needs twice as much, the trapezoid and triangle representation requires almost three times as much, and the heterogeneous representation needs almost four times as much storage space as the identical representation.

Within the next section we will carefully evaluate the performance of spatial query processing based on different decomposition techniques and we will compare it to the identity representation.

6.3 Comparison of decomposition techniques with respect to spatial query processing

In this part of the comparison, we would like to empirically evaluate which object representation technique (undecomposed, convex polygons, trapezoids, etc.) leads to best performance in spatial query processing.

As described in section 1 and in more detail in section 5, spatial query processing consists of two steps, the filter step and the refinement step. According to this two phases, we evaluated query performance of different types of spatial object decompositions comparing their results to the undecomposed object representation.

The performance of the filter step is considerably determined by the performance of the SAM handling bounding boxes. We used the to our knowledge best access method handling bounding boxes, the R*-tree [BKSS 90] with page capacity (directory and data) bound to 2K. The most time consuming operations during queries in the R*-tree are accesses to secondary storage and comparisons within the directory and the data pages. Thus, we counted the number of page accesses and the number of comparisons in directory as well as data pages and then multiplied them by typical time constants.

The refinement step performes computational geometry algorithms for those candidate objects supplied by the filter step. In our bookkeeping of query time we include the time spent with the main memory search structure for object identifiers in the time for the refinement step. This is due to the fact that decomposition techniques simplify complex computational geometry algorithms by using a set of object components. Therefore, we assign the task of handling this redundant object representation to the refinement step. Consequently, the refinement performance is determined by the time spent for computational geometry algorithms and, in case of redundant object representations, the time needed for performing insertions and search operations in the main memory search structure managing the identifieres of found objects. As these performance parameters strongly depend on the particular set of data, we explicitely measured them using implementations of the different decomposition techniques.

Finally, we added the performance parameters of the filter and the refinement step to obtain a measure for the overall query performance of redundant object representations for different spatial queries.

The queries that we performed on the different object representations are classified into point queries (window query with zero extension) and window queries with different window sizes referring to a varying selectivity of spatial queries. The size of the query window was fixed to the values of 0.01%, 0.1%, 1%, and 10% of the data space size (which we consider for typical within real applications). More complex queries, e.g. region queries using SPH shaped query regions, were not considered in this test. Those queries typically are evaluated by performing a window query with a minimum bounding box of the query region followed by more complex computational geometry algorithms on the objects supplied by the filter step of the window query. Therefore, those algorithms are even more crucial in overall query processing and, in fact, additionally favour object decomposition techniques.

The query results are presented in three figures for each of the data files introduced in the last section. The figures depict the following information: the horizontal axis represents the size of the query window. The number below the window size is the percentage of answers with respect to the total number of objects. Let us mention that a low percentage of answers corresponds to a high selectivity and vice versa. The vertical axis gives the time requirements for performing those queries of varying window sizes. The time is given in microseconds per found object. Every decomposition technique is characterized by its own curve carrying an abbreviation of the name of the technique. The first figure corresponds to the time spent for the filter step. The

next one depicts the time needed for performing the computational geometry algorithms on the object/component representations and, in case of redundant object representations, the time spent for the main memory search structure avoiding duplicate refinement operations. Finally, the rightmost figure represents the complete result adding the results of the first two figures and therefore corresponds to the overall query time of spatial query processing.

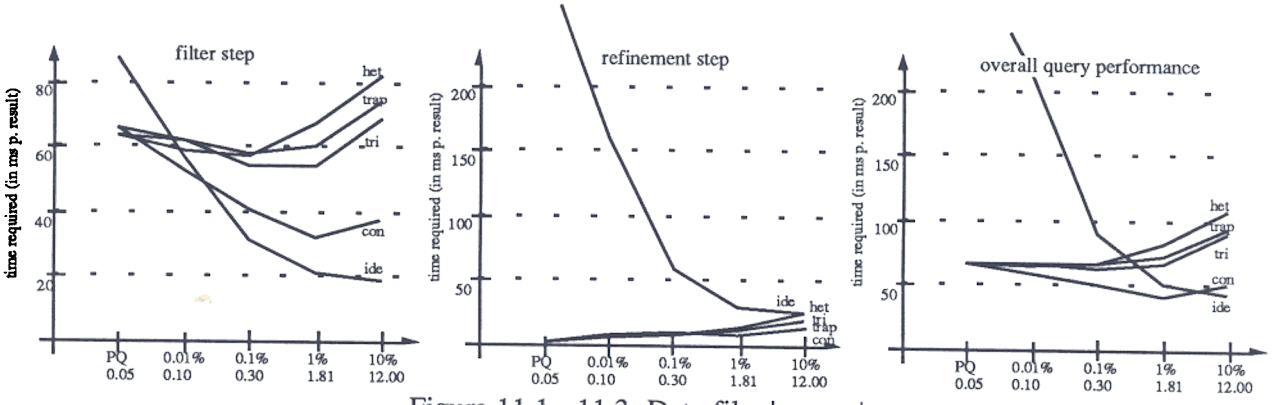

Figure 11.1 - 11.3: Data file: 'europe'

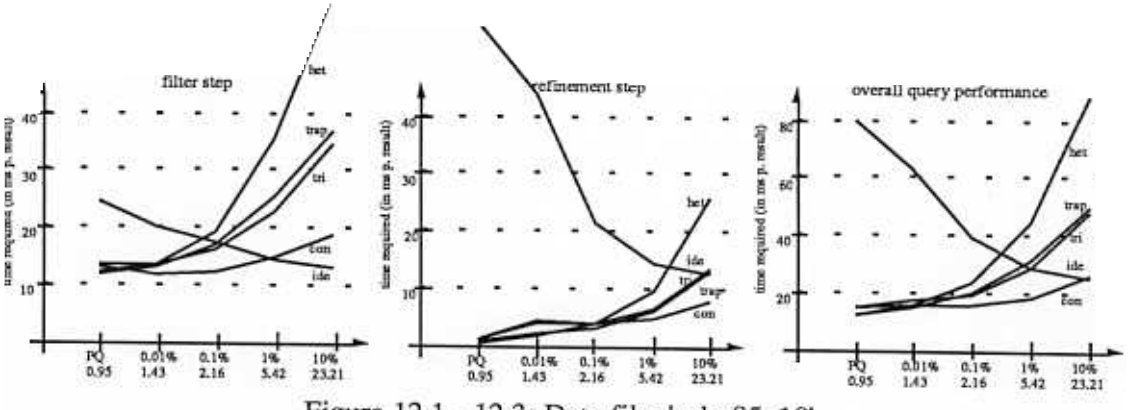

Figure 12.1 - 12.3: Data file: 'sph_85_10'

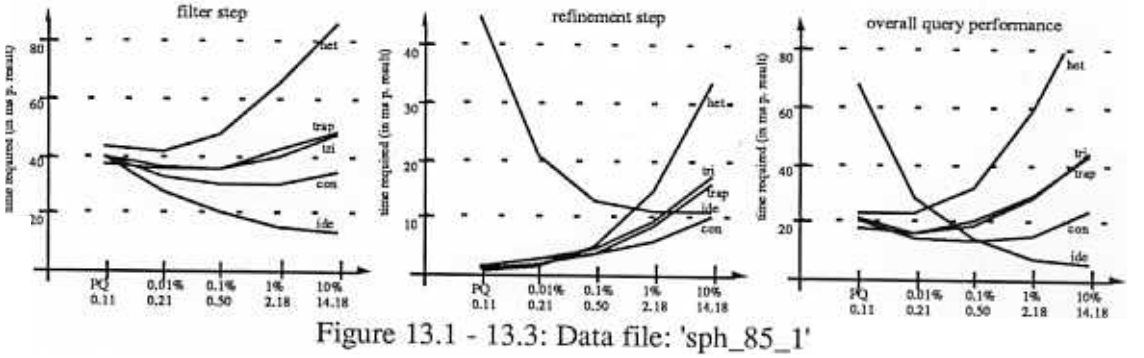

Figure 13.1 - 13.3: Data file: 'sph_85_1'

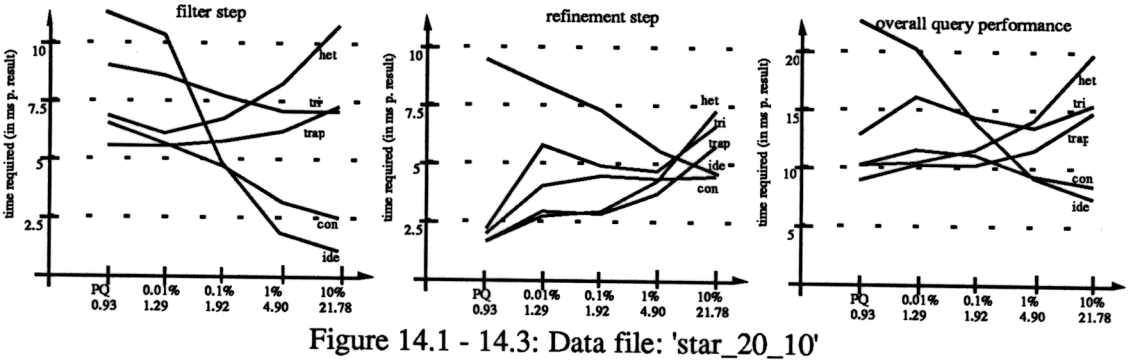

Figure 14.1 - 14.3: Data file: 'star_20_10'

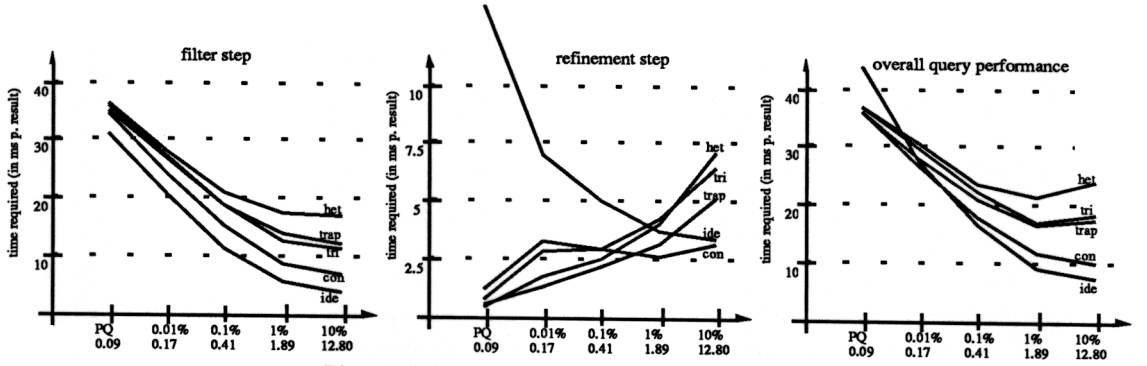

Figure 15.1 - 15.3: Data file: 'star_20_1'

The overall query time clearly shows a strong dependance on the size of the query window. Further important criteria which influence the performance of different object representations are the object complexity, i.e. the number of vertices, and the cover of the objects. The main results are:

- decomposition techniques clearly outperform (up to an order of magnitude) the undecomposed representation for point queries and window queries with small query windows and thus high selectivity
- with decreasing query selectivity the performance of the undecomposed representation improves relative to the performance of decomposition techniques with a break-even point for a rather low selectivity
- the performance of decomposition techniques for window queries of low selectivity strongly depends on the amount of redundancy
- the more complex the objects, the more clearly appears the trends outlined above

Independent of the data file, point queries and high selectivity window queries are performed more efficiently by any of the decomposed object representations than by the undecomposed representation. Particularly, the real data file 'europe' (fig. 11) shows a significant gain of decomposed representations up to the factor 5 in overall query time. As the first two figures, e.g. of the 'europe'-file (fig. 11.1 and 11.2), show, this behaviour is due to the very expensive refinement operations for the undecomposed objects and to the fact, that small query windows will profit from the selectivity of the spatial access method, limiting the number of disk accesses and supplying a small amount of redundant components. Particularly, for the point query, caused by the good container approximation of most decomposition methods, frequently no redundant components are accessed at all. This trend obviously holds for any of the data files. The gain in efficiency of object decompositions depends on the complexity of the data objects. The higher the complexity of the objects with respect to the number of vertices, the better is the performance of object decompositions relative to the identity representation.

Regarding low selectivity window queries, i.e. query windows of 5-10 % of the data space, this trend turns around. Redundant object representations perform worse for large window queries depending on the amount of redundancy. This is strongly caused by the large amount of components which have to be managed by the access method. As queries of small selectivity relate to large portions of the data, the total amount of stored data essentially determines the number of disk accesses necessary to answer those queries. Typically, the amount of stored data for object decompositions is significantly higher than for an undecomposed object representation, as we see from section 6.2, table 2. Therefore, for decomposed object representations the time spent with the access method increases with a growing size of the query window, i.e. a shrinking selectivity of the query (see first figures of the performance results). The amount of redundancy that has to be handled within the refinement step obviously increases, at the same time. Therefore, eliminating redundancy strongly determines the performance of the refinement step of decomposed representations. Contrarily, the average time spent for one explicite object test of the undecomposed representation decreases, as the number of cheap object tests increases (i.e. the bounding box of an object is fully included in the query window and therefore no further action is necessary).

The influence of an increasing object complexity is reflected in an intensification of the basic trends. On one hand, complex objects require complex computational geometry algorithms in the case of undecomposed representation (see fig. 11.2, 12.2, and 13.2), on the other hand, decompositions of complex objects lack in a further increase of redundancy (fig. 11.1, 12.1, and 13.1). Therefore, the performance of high selectivity queries which strongly depend on efficient refinement operations will decrease with an increasing object complexity for the undecomposed object representation. As low selectivity queries basically depend on the number of disk accesses and the amount of redundancy to be handled, object decompositions will degenerate for very complex objects which enlarge the degree of redundancy. (see fig. 11.3, 12.3, and 13.3)

The cover of the objects directly corresponds to the selectivity of a query of fixed size. When only the cover of the objects increases (all other parameters remain fixed), the number of answers will grow. Therefore, the undecomposed representation needs a higher number of expensive refinement operations which can be considered as a penalty for the performance. However, object decompositions suffer in handling a large amount of redundancy in the case of low selectivity queries. (see fig. 12 and 14)

Summarizing over all shapes of objects, decomposition techniques gain by performing cheap computational geometry algorithms in the refinement step. This advantage is strongly valid for high selectivity queries where secondary storage accesses are not dominant. Low selectivity queries, however, supply a high amount of redundancy which must be accessed on secondary storage and handled in main memory. These efforts rise with an increasing size of query windows, i.e. decreasing selectivity. The degree of redundancy of a decomposition method, i.e. the number of components for originally one spatial object, is mainly reflected in the performance of low selectivity queries. The performance of the different decomposition techniques is similar for high selectivity queries. This is due to the fact, that computational geometry algorithms perform very similiar for all decomposition techniques considered here. The undecomposed representation performes worse for high selectivity queries. Depending on the decomposition technique and the particular type of objects, a specific size of query window exists, where the performance curves of the undecomposed representation and object decompositions intersect in a break-even point. Queries with higher selectivity are more efficiently performed by the decomposed representations, queries with lower selectivity are handled faster by the undecomposed representation.

Among the decomposition techniques the convex object decomposition turns out to be the best performing technique. It gains from a relatively small amount of redundancy (see section 6.2, table 2) and very cheap computational geometry algorithms, because the average number of vertices is between 4 and 5. The break-even point of the performance curve, particularly for complex data (fig. 12.3) and real data (fig. 11.3), corresponds to a considerable large query size, i.e. a considerably low query selectivity. For most queries and arbitrary types of objects, the convex decomposition technique performes better than the undecomposed representation.

From our tests we learned, that an optimal decomposition technique is not obtained by minimizing the complexity of the components due to the penalty of a very large amount of redundancy (see 'heterogeneous decomposition'). It is desirable to find a decomposition method which restricts the computational effort for its components and as well reduces the number of components. The performance of such an ideal decomposition method within our test bed will correspond to a parallel to the axis representing query sizes, i.e. its performance will be the same for small and for large query windows.

7 Conclusions

In this paper, we presented a two-level, multi-representation query processing technique for two-dimensional polygonal objects consisting of a filter and a refinement step. The efficiency of the spatial query processor is gained by decomposition of complex spatial objects into simple spatial components and the application of efficient and robust spatial access methods for simple spatial objects. We introduced three new decomposition

techniques for simple polygons with holes. Within an extensive performance comparison, we compared these techniques to each other, to another technique known from the literature and to the undecomposed representation with respect to their performance in spatial query processing. The main results are:

- Decomposition techniques generally perform extremely good for high selectivity spatial queries and they outperform the traditional object representation up to one order of magnitude.
- Decomposition is a proper representation scheme especially for complex objects, i.e. polygonal objects with an average vertex number greater than 80.
- Decomposed representations lead to a very good query performance specially for files with a high cover, e.g. multi-layer maps.
- The convex decomposition turns out to be the best compromise between simple computational geometry algorithms and a moderate degree of redundancy and is the winner of all decomposition techniques.

Summarizing, we can state that query processing based on object decomposition is a promising approach worth to be further researched. In our future work, we will extend the query processor by more complex queries, e.g. enclosure and containment queries. Furthermore, we plan to integrate additional decomposition techniques and other spatial access methods, e.g. the cell tree [Gue 88] and the P-tree [KS 91], into our query processor for an even more extensive comparison.

Acknowledgement

We thankfully acknowledge receiving real data representing the countries of the European Community by the 'Statistical Office of the European Communities'. In particular, we would like to thank Thomas Brinkhoff for preprocessing the above data and converting it to the format required by our SPH files.

References

[AA 83] Asano, Ta. & Te. Asano, 'Minimum Partition of Polygonal Regions into Trapezoids', in Proc. 24th IEEE Annual Symposium on Foundations of Computer Science, 233-241, 1983.

[BKSS 90] Beckmann, N., H.P. Kriegel, R. Schneider & B. Seeger, 'The R*-tree: An efficient and robust access method for points and rectangles', in Proc. 1990 ACM SIGMOD International Conference on Management of Data, 322-331, Atlantic City, USA, May 1990.

[Bur 87] Burrough, P.A., Principles of Geographical Information Systems for Land Resource Assessment', Clarendon Press, Oxford, 1987

[CD 85] Chazelle, B. & D.P. Dobkin, 'Optimal Convex Decompositions', Computational Geometry, G.T. Toissant (Ed.), Amsterdam, The Netherlands: North Holland, 63-134, 1985

[Del 34] Delaunay, B.,'Sur la sphere vide', Izvestiya Akademii Nauk SSSR, VII Seria, Otdeline Matematicheskii i Estestvennyka Nauk, 7, 6, 793-800, 1934.

[DHH 90] Droste, U., H.-G. Harms, H. Horn, 'Decomposition Based Representation and Query Processing of Polygonal Object in Database Systems', Master Thesis (in German), University of Bremen, 1990

[Fre 87] Freeston, M., 'The BANG file: a new kind of grid file', Proc. ACM SIGMOD Int. Conf. on Management of Data, 260-269, 1987

[Gue 89] Günther, O., 'The design of the cell tree: an object-oriented index structure for geometric databases', in Proc. IEEE 5th Int. Conf. on Data Engineering, 598-605, Los Angeles, 1989.

[Gut 84] Guttman A., 'R-trees: a dynamic index structure for spatial searching', in Proc. ACM SIGMOD Int. Conf. on Management of Data, 47-57, June 1984.

[KHHSS 91] Kriegel, H.P., P. Heep, S. Heep, M. Schiwietz & R. Schneider, 'A Flexible and Extensible Index Manager for Spatial Database Systems', submitted for publication, 1991.

[KHS 90] Kriegel, H.P., P. Heep & R. Schneider, 'Design and Implementation of a triangle decomposition for simple polygons with holes', Technical Report, University of Bremen, Germany, 1990.

[KS 85] Keil, J.M. & J.R. Sack, 'Minimum decomposition of polygonal objects', Computational Geometry, G.T. Toissant (Ed.), Amsterdam, The Netherlands: North Holland, 197-216, 1985

[KS 88] Kriegel, H.P. & B. Seeger., 'PLOP-Hashing: A Grid File without directory', Proc. 4th Int. Conf. on Data Engeneering, 369-376, 1988

[KS 91] Kriegel, H.P.& M. Schiwietz, 'The P-tree: an efficient access method for complex spatial objects', University of Bremen, Germany, 1991, in preparation.

[KSSS 89] Kriegel, H.P., M. Schiwietz, R. Schneider & B.Seeger, 'Performance Comparison of Point and Spatial Access Methods', in Proc. "Symposium on the Design and Implementation of Large Spatial Databases", 89-114, Santa Barbara, USA, July 1989.

[NHS 84] Nievergelt J., H. Hinterberger & K.C. Sevcik: 'The grid file: an adaptable, symmetric multikey file structure', ACM Trans. on Database Systems, Vol. 9, 1, 38-71, 1984.

[OM 86] Orenstein, J.A. & F.A. Manola, 'Spatial Data Modeling and Query Processing in PROBE', Technical Report CCA-86-05, Xerox Advanced Information Technology Devision, 1986.

[Oos 90] Oosterom, P.J.M., 'Reactive Data Structures for Geographic Information Systems', PhD-thesis, Department of Computer Science at Leiden University, The Netherlands, 1990.

[Oren 89] Orenstein, J.A., 'Redundancy in Spatial Databases', in Proc. 1989 ACM SIGMOD International Conference on Management of Data, 294-305, Portland, USA, June 1989.

[PS 85] Preparata, F.P. & M.I. Shamos, 'Computational Geometry: An Introduction', Springer-Verlag, New York, 1985

[Sam 84] Samet, H., 'The Quadtree and Related Hierarchical Data Structures', ACM Computing Surveys, Vol. 16, No. 2, 187-260, 1984

[See 90] Seeger, B., 'Design and Implementation of multidimensional access methods' (in German), PhD-thesis, University of Bremen, Germany, 1990.

[SK 88] Seeger, B. & H.P. Kriegel, 'Techniques for design and implementation of efficient spatial access methods', in Proc. 14th Int. Conf. on Very Large Databases, 360-371, Los Angeles, USA, 1988.

[SK 90] Seeger, B. & H.P. Kriegel, 'The Buddy Tree: An Efficient and Robust Access Method for Spatial Databases', in Proc. 16th Int. Conf. on Very Large Data Bases, Brisbane, Australia, August 1990.

[SV 89] Scholl, M. & A. Voisard, 'Thematic Map Modelling', in Proc. "Symposium on the Design and Implementation of Large Spatial Databases", 167-190, Santa Barbara, USA, July 1989.