Published in: Proceedings, 1997 IEEE Symposium on Visual Languages, Capri, Italy, Sep. 23-26, IEEE Computer Society Press, 1997, pp. 197-204.

# Querying GIS with Animated Spatial Sketches

**Volker Haarslev and Michael Wessel**
University of Hamburg, Computer Science Department,
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
http://kogs-www.informatik.uni-hamburg.de/~haarslev/

## Abstract

*We present the design of the visual query system VISCO that offers a sketch-based query language for defining approximate spatial constellations of objects. VISCO smoothly integrates geometrical and topological querying with deductive spatial reasoning. It is based on a strong physical metaphor visualizing semantics of query elements. Approximate queries rely on combined topological and geometrical constraints enhanced with relaxations and "don't cares" that are visualized through live animations.*

**Keywords**— visual query systems, visual parsing, deductive GIS, constraints.

## 1 Motivation and Introduction

The need to develop new interface paradigms for interacting with spatial information systems or databases, especially geographical information systems (GIS), has already been noted elsewhere [1, 2]. With respect to HCI an interface should be convenient, easy-to-use, and actively supporting the user. A strong metaphor can motivate users and guide them through the interaction with a system. In response to these considerations we present the visual query system VISCO (Vivid Spatial Constellations) that provides a sketch-based query language for defining approximate spatial constellations of objects. Our query language can express geometric as well as topological constraints. The user is actively supported by animations of queries that visualize examples of approximate constellations. The query language elements are visualized with the help of a naive physics metaphor utilizing rubber bands, (cross)beams, swivel joints, nails, marbles, etc. The meaning of VISCO's language elements is immediately graspable from the physical properties of their visualizations, e.g. a rubber band may be stretched, shrunk and wrapped around in contrast to a (rigid) beam, a marble can roll around and change its position in contrast to a nail.

VISCO offers several novel features that correspond to issues mentioned in a recent survey on visual query systems for databases. Catarci et al. [3] conclude this excellent survey with a list of most significant issues for the design of next generation visual query systems. VISCO's features incorporate solutions for several of these issues.

- Animation is an essential part of VISCO and illustrates possible variations in user sketches.
- VISCO deals with spatial data types such as points, segments, polylines, polygons and their possible spatial relationships.
- A formal semantics for VISCO is based on a space box (SBox). This SBox [4] combines a qualitative spatial calculus with a description logic (DL) (see also [5, 6] for an introduction to DL) and is grounded on quantitative reasoning (i.e. computational geometry). This logic is an extension of a framework that has been successfully applied to the specification of visual languages [7, 5, 6]. VISCO's query language can be mapped to expressions of this SBox language.
- Data is encoded with the help of the SBox that offers well-defined mechanisms for inferring implicit (spatial) knowledge (e.g. from stored maps).
- DL (as basic part of the SBox) is well suited for expressing incomplete or indefinite knowledge and for dealing with metaknowledge. For instance, DL systems automatically compute the subsumption relationship between DL expressions. The resulting taxonomy can easily be utilized for query optimization.
- VISCO offers powerful tools for approximate questioning that may be used for formulating queries about approximate spatial constellations of database objects.

In contrast to other relevant work [2] that focuses on topological descriptions we adopt a bottom-up approach and parse the sketches and their geometry as drawn by the user. VISCO takes the geometry of query sketches seriously but supports the annotation of meta information which can be used to specify almost pure topological queries. The user may add meta information to a sketched query. This meta information specifies relaxations, additional constraints or "don't cares" that define the interpretation of the query. The visibility of user-defined relaxations and "don't cares" is a major advantage of our approach. In our opinion this explicit meta information (which has to be supplied by the user) is important since drawings are always in a sense "overspecified" and their (relaxed) interpretation strongly depends on the application domain. The effects of relaxations are visualized through animations.

## 2 VISCO: Vivid Spatial Constellations

The interpretation of a user's spatial query is a critical part for any spatial query system. The user's intuition about the interpretation should match with the system's implemented algorithms. For instance, the concept of a right angle has a strong significance in a CAD system but must be relaxed in a GIS system. This shows that the "correct" interpretation depends on the actual application domain. Thus, VISCO offers tools that specify meta information resembling the user's idea of the interpretation. This approach demands more skill from the user but makes the intended interpretation explicit. This is the reason why VISCO's user interface (that is derived from GenEd [6]) offers active support through animations.

### 2.1 Language Elements

The building blocks of VISCO are divided into meta objects and query objects. *Query objects* consist of points, segments, polylines, or polygons. They represent the database objects to be retrieved and can be either interactively sketched or computed (so-called *derived* objects) with the help of VISCO's operators (see Section 2.3). Most meta objects are generated with the help of VISCO's interface. However, we like to emphasize that spatial relationships between query and/or meta objects are only computed by parsing the query. The system ensures that distinct individual objects can always be distinguished visually (by using color etc).

We try to reduce the burden of the user's intuition about VISCO's language elements with the help of a strong metaphor. This metaphor is based on common sense physical semantics. We attach to query objects the properties of real-world objects. Thus query objects resemble marbles, nails, swivel joints, rubber bands, and (wooden) beams (illustrated in Figure 1). Their physical properties add extra semantics to VISCO's language elements. Query objects may be further constrained to match with DL concepts derived from database objects. These concepts denote the functionality of database objects in the real world (e.g. for maps: points as subway stations; streets as polylines; lakes, parks as polygons, etc). Meta objects serve different purposes. Thus most of them relax or enforce some spatial constraints and guide the query interpretation process. Supported *meta objects* are transparency films, enclosures, compasses and arrows, upper- and lower scaling rectangles, guiding lines, and (constrained) crossbeams.

#### 2.1.1 Transparency Films

Keeping this metaphor in mind the applicability of VISCO's language elements is easily explained. The basic building block is a *transparency film* (of an overhead projector). Every transparency has its own local cartesian coordinate system and a rectangular shape. Users can interactively start drawing query language elements upon a transparency. A collection of drawn elements defines a (sub)constellation with relevant geometrical/topological relationships. The size of a transparency and the size and position of elements drawn on a transparency are taken seriously and do matter. Transparencies can be scaled, translated, rotated and

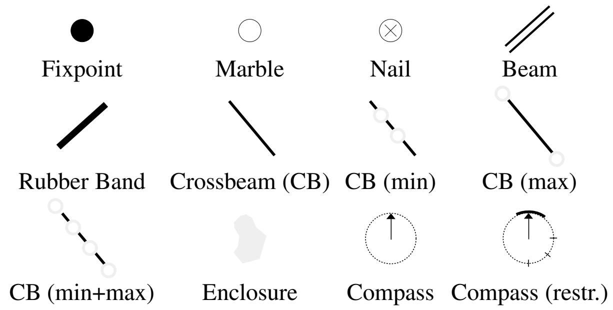



Figure 1: Basic language elements of VISCO

stacked up like layers. Transparencies always have a fixpoint (with respect to transformations) which can be any nail (isolated or as vertex) on the transparency.

Figure 2 illustrates various examples. Figure 2a shows a simple unscalable transparency with its fixpoint in the center. The transparencies in Figure 2b-f may be scaled as follows: only vertically down (b), any direction (c), only proportionally (visualized by dashed guiding lines for the vertices) in any direction (d), only proportionally down with an uncentered fixpoint (e), only proportionally up (f). The transparencies in Figure 2g-j also constrain the upper and/or lower limit of scaling (visualized by dashed horizontal or vertical lines): proportionally (g) or arbitrary (h) with lower and upper limit, only vertically up with upper limit (i), only vertically with upper and lower limit (j). In Figure 2k-l we additionally allow rotation of the transparencies around their fixpoint. This is specified by the compass disk with the fixpoint as center and an arrow as hand. The compass in Figure 2k allows free rotation around the fixpoint, while that in Figure 2l constrains the possible rotation to an angle interval (visualized as bold arc). In general, a compass may constrain rotation to angle intervals and discrete angle values (see Figure 1). A compass allowing only one discrete angle may be abbreviated as a single arrow (its hand) in order to avoid visual clutter.

#### 2.1.2 Enclosures and Points

Imagine, we sketch an enclosure on a transparency. An enclosure is a simple polygon (with optional holes) whose boundary has to be a closed and not self-intersecting polyline. An *enclosure* is a meta object adding to its denoted area the semantics that all enclosed objects must stay inside of this (fenced) area. Also, it specifies that the positions of some of its enclosed points have to be relaxed: a point drawn inside an enclosure is by default a *marble* that can move around but has to stay inside. However, a *nail* cannot change its position (even inside of an enclosure). Enclosures can be *translucent* or *opaque* and are displayed with a gray texture. The associated semantics is described below (see Section 2.2). It is also possible to generate so-called ε-*enclosures* that are computed by operators.

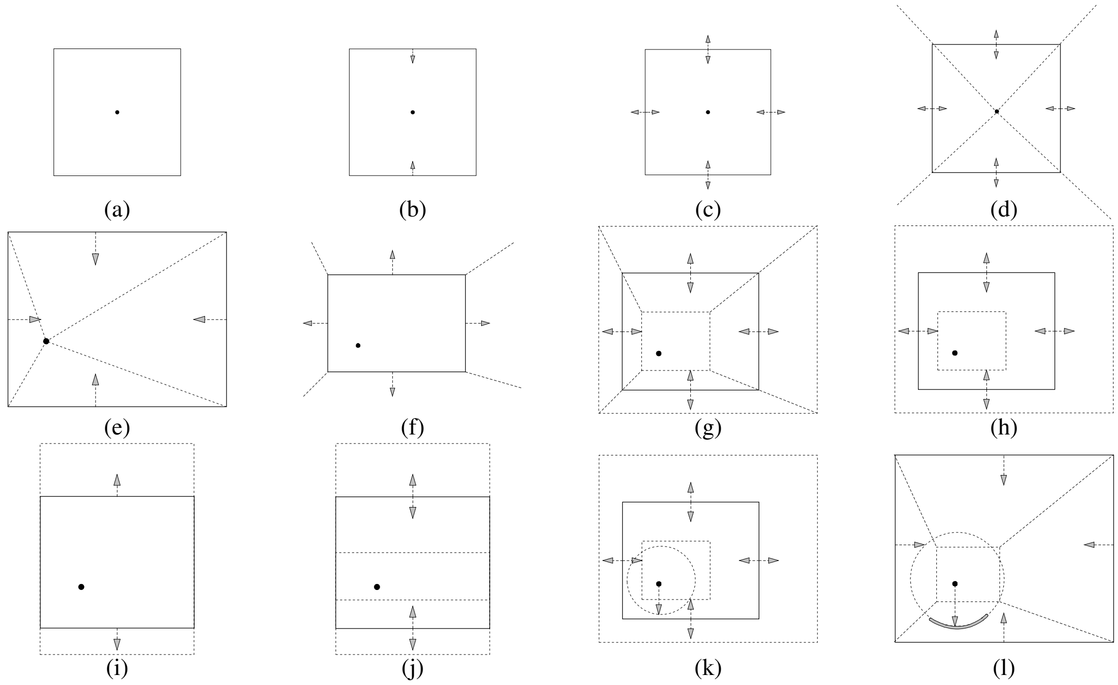



Figure 2: Various applications of transparency films

#### 2.1.3 Line Segments, Polylines and Polygons

Another basic language element is an (undirected) line *segment* with its two end points. Each point can be either a marble or a nail. It should be noted that all query objects except marbles and nails are compound objects, which have component objects. These components are itself "first class" objects. Segments or edges (as components of polylines) act either as *beams* of fixed length or stretchable/shrinkable *(atomic) rubber bands*. A beam matches a line segment of fixed length (as specified) in the database. A rubber band represents a topological structure that matches an arbitrary polyline in the database. However, an *atomic rubber band* is defined as indivisible and therefore matches a single database line segment. In our physical metaphor it resembles somehow a *telescope antenna.* There are three types available: the $\leq$ ($\geq$) atomic rubber band represents a segment with a given maximal (minimal) length, and the "don't care" $\star$ atomic rubber band does not enforce any constraints on the length of segment. Vertices (points) connect edges and also play the role of *swivel joints* that are either marbles or nails. Polylines may be closed but not self-intersecting. Regions are defined as simple polygons with polylines as boundaries.

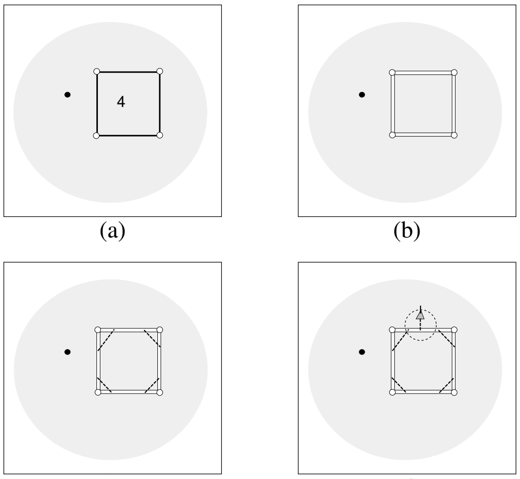
Enclosures affect directly the position of points and indirectly the position and/or shape of polylines (and thus polygons). Form variability of polylines and polygons is achieved by position (length) variability of their vertices (edges). Furthermore, number constraints can be associated with some query objects. An *at most constraint* for a rubber band, polygon or polyline specifies the maximal number of line segments of the database object that is matched against this query object. The number of segments of a query polygon or polyline always forms an implicit at least constraint. An atomic $\star$ rubber band is therefore equivalent to a rubber band with an 'at most one' constraint.

For instance, Figure 3a shows a transparency containing the specification of an arbitrary quadrilateral whose edges are rubber bands and whose vertices are marbles that can float inside of the enclosure (gray circle). If we discarded the *at most four* constraint for the polygon, this query would retrieve arbitrary polygons (with at least four line segments). Alternatively we could have defined the quadrilateral without the at most constraint be using $\star$ *atomic* rubber bands instead of rubber bands. Figure 3b-d demonstrates various other definitions: arbitrary quadrilateral with edges of fixed length (b), floating square of fixed shape (c), floating square of fixed shape and upright orientation (d).

The examples in Figure 3c-d use another meta object: a *crossbeam* constrains the angle between two connected edges. A simple crossbeam freezes the angle as drawn. This can be relaxed with minimal and maximal angles indicated by bullets lying on the crossbeam (see Figure 1).

### 2.2 Relationships between Objects

At first it has to be noted that the user's query is constructed step-by-step using VISCO's interactive environment. Our




(a) (b) (c) (d)

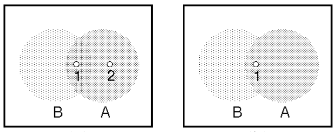
Figure 3: Various quadrilaterals

language is not designed to be visually complete because the construction history of the query affects its semantics. Parsing a VISCO query is a progressive process as the construction continues. However, at each construction step WYSIWYG is taken seriously. Whenever a new query object is introduced, various relationships to other (partially) *visible* objects are calculated. For instance, the ‘contains’ relationship (or constraint) is enforced between enclosures and their enclosed query objects. Components of complex query objects are considered as individual “first class” objects (e.g. the segments of a polygon). So, any discussion of query objects applies to component objects as well.

We already mentioned that enclosures can be translucent or opaque. An opaque, overlapping enclosure stacked in front can (partially) hide some underlying enclosures and also objects already drawn. If the topmost enclosure is translucent it will not hide the underlying elements. Any visible elements stacked below can still be addressed since they are not hidden (WYSIWYG). Thus, we can enforce an object to be inside various (underlying) enclosures, if they are not hidden. This is an explicit ‘and’ constraint holding for all visible ‘inside’ relationships of this object. See Figure 4a-b: in case a), marble 1 has to be inside $A$ and $B$, marble 2 has to be inside $A$. In case b), marble 1 only has to be inside $A$.

If an object $Y$ is (partially or fully, see below) visible (with respect to occluding enclosures) at the time of creation of a query object $X$, we enforce some additional high-level spatial constraints between $X$ and $Y$. These enforced spatial relations strongly affect the semantics of a query. The following relations (and their unlisted inverses) are recognized by VISCO:

- **Enclosure $\times$ query object:** *contains*
- **Point $\times$ segment:** either *intersects* (if the point lies on the line segment) or *disjoint*




(a) (b)

Figure 4: Translucent and opaque enclosures

- **Segment $\times$ segment:** either *intersects* (this includes any constellation where two segments have at least one common intersection point) or *disjoint*.

There are no point $\times$ point relations, because the user interface does not allow the definition of two points at the same position. Two points are therefore always disjoint. A common vertex of e.g. two line segments of a polygon is assumed to be only one unique point object (and not two objects that are equal). This also implies that there exist no congruent query objects.

The *intersects* and *disjoint* relations are computed between points and lines, and lines and lines regardless whether the point or line objects are components (because of the “first classness” of component objects). It can be easily shown –by taking into account all component point-line relations– that the relations *intersects* and *disjoint* are expressive enough to define, for example, all interesting line-line relations (e.g. disjoint, crosses, touches, meets, overlaps, contains, inside, covers, covered_by).

However, no relation between a compound object and any of its components will be introduced. As an example, if we have two individual line segments that are crossing each other and completely visible with their two endpoints, the system will calculate 10 relation constraints (2 line $\times$ line tuples, 4 point $\times$ line tuples, 4 line $\times$ point tuples).

We also support relaxations of spatial relations. Relaxations can be achieved by abstracting from the identity of component objects and regarding them as a proxy of their compound object. As an example, the description of a query where a segment crosses a polygon might include the constraint $crosses(segment_1, poly_2\_segment_5)$. If we abstract from the unique identity of $poly_2\_segment_5$, we can rewrite this constraint to $crosses(segment_1, poly_2)$ and could get more matches. VISCO supports these relaxations through a graphical annotation method which is not discussed here due to lack of space.

Generally speaking, WYSIWYG determines whether a *disjoint* or *intersects* relation is enforced. The visual information present at creation time has to unambiguously imply the relation constraint. For instance, if an object $A$ is partially hidden by an enclosure containing another object $B$, one cannot visually decide at the time of creation of object $B$ whether $A$ and $B$ are disjoint or not, and the visually present information does not accidentally imply ambiguous relationships (see Figure 5).

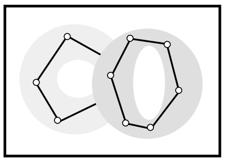


Figure 5: Unknown spatial relations between two objects in different enclosures (disjoint or touching or overlapping?)

The concept of an enclosure can also be applied to stacked transparencies and their fixpoints. The fixpoint of a transparency *A* can play the role of a marble enclosed on a lower-level transparency *B*. This implies that *A*'s fixpoint can float inside of the enclosure on *B* (and thus the flotation of the whole transparency *A* is constrained). Note that the whole coordinate system of *A* is affected by *B*'s actual transformation. Enclosures for fixpoints of other transparencies must be opaque. We restrict the visibility of fixpoints to at most one transparency on a lower-level, i.e. a hierarchy of stacked transparencies may result. The fixpoint of the top-level transparency (in the hierarchy) is implicitly constrained to be inside a "whole world area". See Figure 6 for an example: this query defines a rectangle of fixed size and shape that touches a scalable circle at the right.

### 2.3 Operators and Computed (Derived) Objects

VISCO's interface supports the application of operators to objects. When a new object is created, its component objects will be introduced as first-class objects. We call this "top-down" creation of objects. Conversely, a new object can be aggregated (what we call "bottom-up" creation) by choosing existing query objects as components of the new object (e.g. choose two existing nails as end points of a new rubber band). Operator-created objects are also first-class (*derived*) objects. For instance, the intersection point of two crossing line segments can be computed. Type and properties of derived objects are automatically computed but can always manually changed by the user through subsequent operator applications. Note that derived objects are visualized in a different color in order to distinguish them from meta and other query objects. An animation of a query is available at any time. It demonstrates possible variations and facilitates visual inspection. Constraints can become redundant during the incremental construction process, e.g. a compass allowing free rotation for a beam that is fixed between two nails is superfluous and will be removed from the query.

Due to lack of space we will not discuss all operators of VISCO in detail. The most important ones are the following (of course, additional operators for changing object positions, parameters etc. are required). Note that we consider nails and marbles as points, and rubber bands, atomic rubber bands, and beams as line segments.

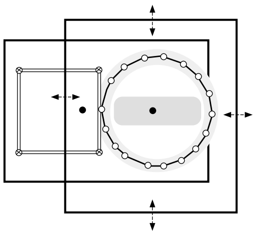


Figure 6: Scalable circle touching a rectangle

- **NIL**: create transparency
- **Transparency:** create query object, create enclosure
- **Enclosure:** toggle enclosure's translucency, negate enclosure (precondition: enclosure has no holes), create at most constraint (specifying the maximal number of found database objects inside this enclosure)
- **Query object:** attach DL concept descriptor (e.g., "road") to object
- **Point:** create (derived) $\varepsilon$-enclosure with radius $r$
- **Nail:** declare as transparency fixpoint, declare as marble (precondition: nail inside an enclosure)
- **Marble:** declare as nail
- **Point $\times$ point:** aggregate line segment from points
- **List of line segments:** aggregate polygon, aggregate polyline
- **Beam, atomic rubber band:** create compass
- **Polygon, polyline:** create compass, then inherit the compass to all beams and atomic rubber band segments
- **Line segment, polyline:** create $\varepsilon$-enclosure with radius $r$, create midpoint
- **Point, line segment, polyline:** declare object as a part of a searched object (e.g. a polyline as part of a polygon boundary)
- **Polygon:** create inner enclosure, create outer enclosure, create $\varepsilon^+$-, $\varepsilon^-$-enclosure (see Figure 7, a radius $r$ is always required), create midpoint
- **Beam:** declare as rubber band, declare as atomic rubber band
- **Atomic rubber band:** change type to $\leq$, $\geq$ or $\star$, declare as beam, declare as rubber band
- **Rubber band:** declare as beam, declare as atomic rubber band, create at most number constraint
- **(Beam $\vee$ atomic rubber band) $\times$ (Beam $\vee$ atomic rubber band):** create crossbeam between them
- **Line segment $\times$ line segment:** create intersection point (precondition: a *crosses* relation). The new point will be a marble (if inside any enclosure), or a nail otherwise.
- **Polygon or polyline with at least one non-atomic rubber band segment:** create at most number constraint

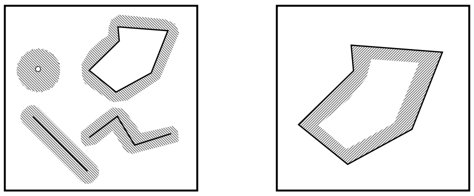



Figure 7: Examples of ε-enclosures

### 2.4 Extended Example: City Maps

In the following we present two extended examples with increasing complexity of their queries. All examples are taken from a ‘city map’ domain. Figure 8 shows a raster image displaying a small subsection of the city of Hamburg, located in Northern Germany. There exists a corresponding vector version of this map that is represented in our spatial database with the help of a space box (SBox). The data elements of the vector map consist of text, points, polylines, and polygons. All data elements have attributes describing their role in the city (e.g. lake, pond, street, church, building, etc). The attribute values correspond to DL concept descriptions organized in a predefined taxonomy. Data elements are represented in our SBox as instances of concept descriptions that combine conceptual and relational (spatial) knowledge.

The first example deals with a constellation where a subway station is in the vicinity of a church. The intended target of the query is displayed as the first element of Figure 9. The other elements show the three construction steps of the corresponding query.

1. Create a transparency with a fixed size of $300 \times 300$ meters. We assume that the subway station and the church are represented as points in our database.
2. Draw a nail on the transparency and attach the predefined concept ‘subway station’. We declare the nail as the fixpoint of this transparency. The transparency itself is unrelated to any other transparency. This implies that the fixpoint may coincide with any point object in the database (i.e. on the map).
3. Generate with a predefined operator a circular ε-enclosure with a radius of 100 meters around the fixpoint. Afterwards, draw a point as a marble inside of the enclosure and attach the concept ‘church’ to the marble.

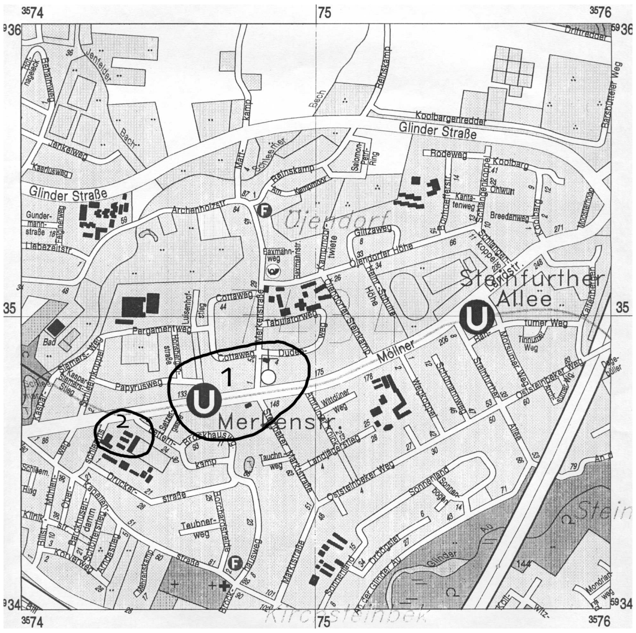



Figure 8: A subsection of the city of Hamburg

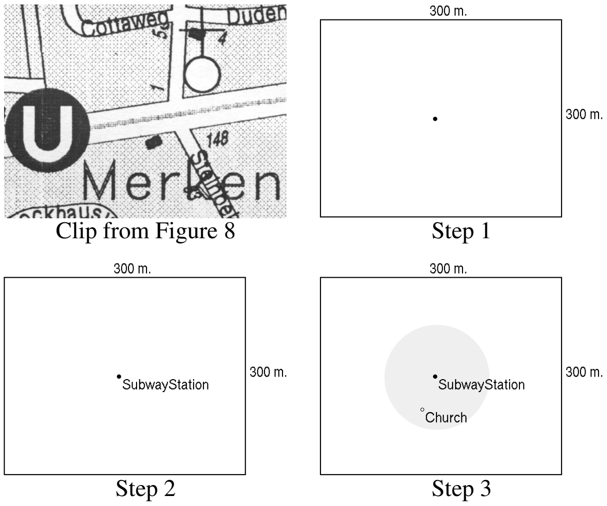



Figure 9: A subway station with a church in its vicinity

The second example describes a constellation where we search for three buildings aligned in parallel. The size of the buildings may vary individually. The first element of Figure 10 shows the intended match for the query while the other elements illustrate the query construction process.

1. Create an arbitrarily scalable and rotatable transparency. Add a large rectangular enclosure to this transparency.
2. Create three rectangles. Their vertices are declared as marbles and their edges as rubber bands. Each rectangle has to match a database polygon with exactly four line segments. Please note that each rectangle defines a quadrilateral (see also Figure 3a). The quadrilaterals are enforced to stay disjoint from one another. We attach the concept ‘building’ to each quadrilateral.
3. Attach to each edge of the quadrilaterals an arrow constraining the edge’s orientation to a single fixed angle (as visualized).

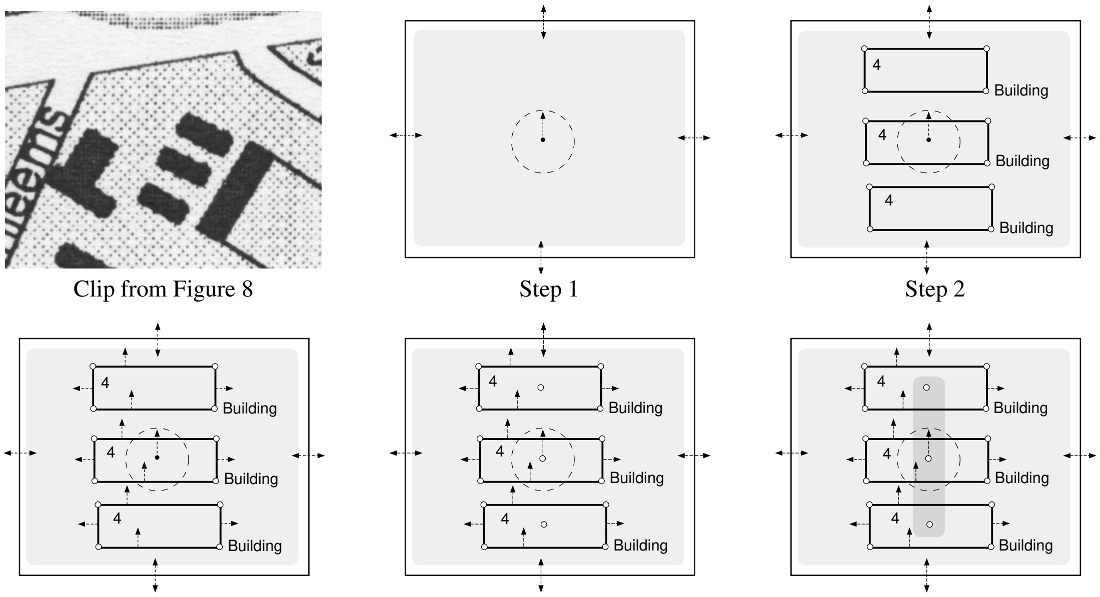



Figure 10: Three adjacent buildings aligned in parallel

4. Apply to every quadrilateral the operator “create center point” resulting in three (derived or computed) marbles.
5. Sketch an enclosure inside of the large enclosure denoting admissible positions for the center points of the buildings. This area defines possible deviations for the alignment.

### 2.5 Formal Semantics with SBox

VISCO’s queries are basically spatial constellations based on topological and geometrical relationships. The spatial parsing of queries is accomplished with a framework that has been successfully applied to the specification of visual languages [7, 6, 5]. This framework has been extended to incorporate topological and geometrical relations as semantically interpreted predicates for deductive spatial reasoning [4]. The framework is implemented as a space box (SBox) extending the semantics of a description logic (DL). The SBox’s semantics is based on computational geometry dealing with 2D polygons.

Due to lack of space we can only roughly sketch out the ideas behind VISCO’s semantics that is explained in detail elsewhere [8]. In general, a query is mapped to a DL concept expression that is processed by our SBox. The SBox provides reasoning services computing instances (i.e. database objects) of DL concepts that fulfill the constraints defined by these concepts. Thus the subsumption between concepts and queries is automatically computed and is available for meta reasoning. For instance, query subsumption immediately supports query optimization.

## 3 Related Work

VISCO can be classified as a visual query system for spatial information systems that uses sketched queries combined with deductive reasoning. A recent and complete survey on visual query systems for database systems handling conventional data can be found in [3]. Other relevant work [9, 2] reviews especially visual query system for spatial information systems. A related approach that also uses spatial relations [10] deals with symbolic descriptions and retrieval in image databases. In the following we shortly review four approaches [9, 11, 12, 2] that come closest to the ideas and concepts behind VISCO. We especially focus our attention on the spatial properties of their query languages.

An iconic query language for GIS is presented in [12]. Icons represent geographic objects such as lakes, rivers, countries, etc. Topological relationships are computed from the icons of a query. It is also possible to specify orientations or circular search areas. The system provides a “foreground” mode which is used to specify relevant relationships between objects. Other accidental relationships are interpreted as “don’t cares”. The user interface is very abstract with simple visualizations. Geometrical aspects cannot be specified. Only a small set of relations without a formal model is allowed.

Cigales [11] is a “query by visual example” system which also allows the user to draw a query. However, queries cannot be sketched but have to be created with the help of operators. Thus, the drawings are only visualizations created by the system. The implied look-and-feel of Cigales’ user interface appears to be quite tedious and modal. Furthermore, the layout of queries can change dramatically after their reformulation and might confuse users.

Sketch [9] was the first language proposing a metaphor for drawing sketches on a blackboard. Sketch allows free-form elements as components of sketches but it strictly divides queries into propositional and spatial conditions. A sketch is parsed and translated to a formula in a corresponding logical calculus. The problem of "don't care" relations is solved by layers that can contain common objects. Topological relationships are only computed for elements of a layer. The language Sketch has formal semantics but the topological relationships have no mathematical foundation. Sketch does not support the integration of geometric properties into queries. It also requires that database objects are "pretyped", i.e. they cannot be recognized through their geometrical properties. For instance, think of a CAD drawing of a transistor that consists of a flat unordered and unstructured "spaghetti" collection of line segments.

Spatial-Query-by-Sketch [2] is distinct to the previous approaches with respect to similarity matches. It uses conceptual neighborhoods of topological relations for relaxation of queries. For instance, a 'touches' relationship between two objects can be very similar to a 'disjoint' or 'intersects' relationship provided the (positive or negative) gap between the objects is below an appropriate threshold. Spatial-Query-by-Sketch allows multi-modal user input for specifying annotations or desired relaxations. It provides no facilities for specifying "don't cares" that apply to relationships between objects. We argue that metric information is existent in many domains and should not be discarded and then subsequently added.

## 4 Conclusion

VISCO is in several aspects distinct to the four approaches mentioned above. It is expressive enough to define geometrical as well as almost pure topological queries or a combination of both. It yields high expressiveness by interpreting topological relations as high-level qualitative constraints enriched with meta information. VISCO offers a powerful but still quite simple physical metaphor for defining queries as spatial constellations. It is possible to specify approximate or vague objects and constellations. Animations are used to visualize vagueness and "don't care" conditions as "vivid spatial constellations". VISCO provides a clear distinction between query and meta objects. The meta information (i.e. additional specifications guiding sketch interpretation) is directly visible to users. There exist no 'hidden' relaxations that might confuse the user's model of query interpretation. A prototype implementation of VISCO will be shortly available for experimental evaluation.

## Acknowledgment

We would like to thank our colleagues Ralf Möller and Bernd Neumann for valuable discussions and thoughtful comments. The maps were generated from data donated by the 'Amt für Geoinformation und Vermessung, Hamburg'.

## References

[1] M.J. Egenhofer, "Why not SQL!", *International Journal on Geographical Information Systems*, vol. 6, no. 2, pp. 71–85, 1992.

[2] M.J. Egenhofer, "Spatial-Query-by-Sketch", In VL'96 [13], pp. 60–67.

[3] T. Catarci, M.F. Costabile, S. Levialdi, and C. Batini, "Visual Query Systems for Databases: A Survey", *Journal of Visual Languages and Computing*, vol. 8, no. 2, pp. 215–260, Apr. 1997.

[4] V. Haarslev and R. Möller, "SBox: A Qualitative Spatial Reasoner –Progress Report–", in *11th International Workshop on Qualitative Reasoning, Cortona, Tuscany, Italy, June 3-6, 1997, Pubblicazioni N. 1036, Istituto di Analisi Numerica C.N.R. Pavia (Italy)*, L. Ironi, Ed., June 1997, pp. 105–113.

[5] V. Haarslev, "A Fully Formalized Theory for Describing Visual Notations", in *Visual Language Theory*, K. Marriott and B. Meyer, Eds. Springer Verlag, Berlin, 1997, In press.

[6] V. Haarslev and M. Wessel, "GenEd – An Editor with Generic Semantics for Formal Reasoning about Visual Notations", In VL'96 [13], pp. 204–211.

[7] V. Haarslev, "Formal Semantics of Visual Languages using Spatial Reasoning", in *1995 IEEE Symposium on Visual Languages, Darmstadt, Germany, Sep. 5-9*. Sept. 1995, pp. 156–163, IEEE Computer Society Press.

[8] V. Haarslev, "The Backstage of VISCO: An Extended Formal Theory for Interpreting VL Queries", Submitted for publication, 1997.

[9] B. Meyer, "Pictorial Deduction in Spatial Information Systems", in *1994 IEEE Symposium on Visual Languages, St. Louis, Missouri, Oct. 4-7*. Oct. 1994, pp. 23–30, IEEE Computer Society Press.

[10] A. Del Bimbo, E. Vicario, and D. Zingoni, "A Spatial Logic for Symbolic Description of Image Contents", *Journal of Visual Languages and Computing*, vol. 5, no. 3, pp. 267–286, Sept. 1994.

[11] D. Calcinelli and M. Mainguenaud, "Cigales, a Visual Query Language for a Geographical Information System: the User Interface", *Journal of Visual Languages and Computing*, vol. 5, no. 2, pp. 113–132, June 1994.

[12] Y.C. Lee and F.L. Chin, "An iconic query language for topological relationships in GIS", *International Journal on Geographical Information Systems*, vol. 9, no. 1, pp. 25–46, 1995.

[13] *1996 IEEE Symposium on Visual Languages, Boulder, Colorado, USA, Sep. 3-6*. IEEE Computer Society Press, Sept. 1996.