Technical Report

Prototype Application Sketcho!

University of Maine

Department of
Spatial Information Science and Engineering
and
National Center for Geographic Information and Analysis

Andreas Blaser

Boardman Hall 321
Orono, Maine 04469, USA
abl@spatial.maine.edu

NCGIA
National Center for Geographic Information and Analysis

Orono, February 22, 1999

TR99-1

Table of Contents

NCGIA
National Center for Geographic Information and Analysis
Technical Report TR99-1
University of Maine, Boardman Hall, Orono, Maine 04469, USA
Andreas Blaser, abl@spatial.maine.edu, Bd Room 321

Prototype Application *Sketcho!*

1. Introduction

This report provides an overview of design and implementation related issues of a sketch-based query user interface for Geo Information Systems (GIS) and its prototype implementation called *Sketcho!*. The prototype application was developed in the scope of the Spatial-Query-by-Sketch project (SQbS), by the NCGIA in Orono and funded in part by the Airforce Laboratories in Rome. Beside the description of the used data structures and data models, we review the sketching process, as well as the different levels of abstraction.

This technical report seeks to explain the applied principles and methods of the prototype, for user specific questions we refer to the *Sketcho!* users guide (Blaser, 1999) and the documentation on the *Sketcho!* CD-ROM.

1.1. Motivation and Goal

Sketching and talking are very common and expressive forms of interaction among people in their day to day life, but when it comes to human-computer interaction, then people have to rely on much more primitive modalities, such as pointing and typing. On the other hand, if computers *could* understand either spoken word or sketched descriptions, an interaction with computers would be much more intuitive and more powerful. We believe that such new multi-modal forms of user-computer interaction are particular helpful within heterogeneous domains, such as GIS or other information systems, because sketching and talking are in many ways more expressive and such more suitable to describe complex data structures.

Sketching in particular seems to be well suited to express spatial relationships and constraints among objects, most of all, because it is a visual and two-dimensional language. Therefore, we believe that a sketch-based user interface has a great potential to query a spatial database.

1.2. Theoretical Foundation and Methodology

The theoretical foundation of the prototype is based on three research areas that have been investigated previous to the implementation.

- User interface design,
- Sketch behavior of people (human subject testing, concerned with sketching habits), and
- Spatial relation theory

The initial step was to build an animated mockup of a sketch-based query user interface using Macromedia Director (Blaser, 1996),(Blaser, 1997). This simulation was a good starting point for developing interaction strategies and testing the applicability of new interaction methods. The conceptual design of a sketch-based application for GIS has been examined in (Egenhofer, 1996; Egenhofer, 1996; Egenhofer, 1997). In order to acquire knowledge about the different real world strategies people use when they are sketching, we conducted an extensive survey involving 34 subjects with different educational and cultural background and analyzed over 90 sketches on an object by object base (~1200 sketched objects). The results have been published in a technical report (Blaser, 1998). The theoretical foundation for the data model of the implementation finally, is rooted in different theories concerned about binary spatial relations and investigations made specifically for this project. These include theories about topology (Egenhofer and Franzosa, 1991; Egenhofer and Mark, 1995), metric (Shariff, 1996) and directions (Goyal and Egenhofer, (in press)).

The implementation, itself was conducted in standard fashion, applying a top down approach, that is the conceptual design of all data objects and classes was done from general to more specific types (classes). Once the overall design of the data model was decided, the functionality of the prototype was added in a sequential manner, from recording and assessing single strokes to the visualization of the formal representation of the sketch.

2. User Interface

The user interface of every device is a central part of any interaction between user and machine, but unfortunately it is exactly this link in the chain that often gets insufficient attention. Hence, many user interfaces in the past were clumsy, unintuitive, difficult to use, or simply inappropriate for what they were designed to. But as processors' clocks aim for gigahertz cycles, the excuse that there is not enough processing power to support sophisticated user interfaces makes less and less sense. Hence, many application developers have begun to pay more attention to the design of user interfaces and such considering the human component more important than in the past.

The design of the user interface that is used for the SQbS prototype is based on a simple sketchpad metaphor. That is the user should have the basic functionality provided by a simple piece of paper, a pencil, and an eraser. This original tool set is to be extended by non-obtrusive features that a computerized environment supports, such as enhanced editing capabilities, multiple views of the same data, analyzing tools, and polymorphous characteristics of the input device. However, this complex functionality should be hidden from the user, for whom an interaction with a sketch-based device should be as natural and intuitive as the scribbling on a notebook.

To achieve this goal we have implemented various visual clues and forms of interaction that people already are familiar with in their everyday life. The readiness of the sketch, for instance, is indicated by lights with different colors (red, yellow, and green) and the tool currently in use is indicated by familiar icons, such as a hand for pointing and selecting, a special gesture for grabbing, and a pen that indicates drawing.

Other visual clues include the appearance of the drawn objects themselves. If an object has been detected as such, then it changes its color. Hence, objects can be distinguished by looking at the drawing alone. There are three stages that can be distinguished like this, the first stage (color blue) indicates that an object is currently being modified, extended, or that it has just been created. Just completed objects—that is an object that has been processed—change their color to green, the indicator for the last processed object. All other objects are black.

However, for certain operations it was necessary to introduce buttons or menu items, but again, we tried to be as intuitive as possible, relying to a great extent on a user's common knowledge. Such, for a normal sketch it is only necessary to tell the application that the sketch is completed and ready to be processed (a single button to be pressed) in order to initiate a database query—no other button has to be pressed. The prototype incorporates three different levels of abstraction that are implemented as views, but only one view is mandatory for every, the other two views are optional.

3. Data Model

The prototype implementation is programmed in an object-oriented environment (Microsoft, 1999) and the data model is object-oriented as well. The programming languages used are C and C++ and the implementation platform is a standard PC with a Microsoft Windows 95/98 operating system. The graphical functions rely on the MFC library (Microsoft Foundation Classes) provided with the Microsoft compiler.

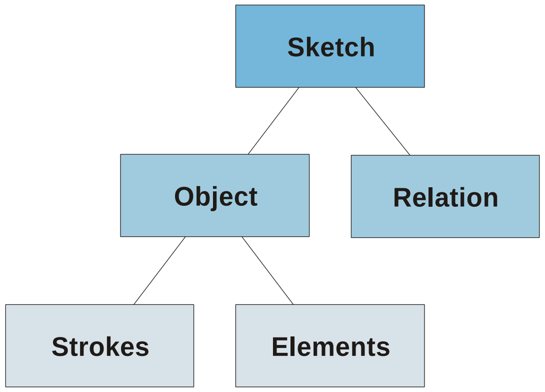


Figure 1 General data structure

The general data model of the prototype application is shown in Figure 1. The sketch is the highest order data object and it consists basically of drawn objects and relations between those objects. Objects on the other hand consist of primitive drawing strokes and sketch elements that are derived from these strokes. The following sections describe links and dependencies between the different mayor classes and their functionality.

3.1. Sketch

The sketch class (CSketchoDoc) is the principal data storage class. Each sketch has only one single sketch object, but the application can have more than one sketch open at the same time.

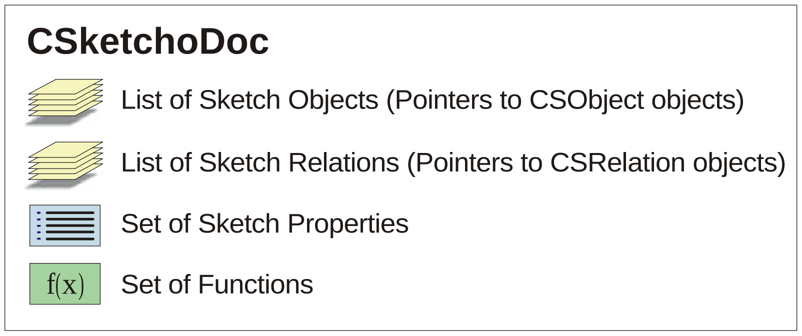

CSketchoDoc
List of Sketch Objects (Pointers to CSObject objects)
List of Sketch Relations (Pointers to CSRelation objects)
Set of Sketch Properties
f(x)
Set of Functions

Figure 2 General overview of the CSketchoDoc class

A sketch object maintains two essential lists: the first list stores pointers to sketch objects (CSObject) and the second list stores pointers to relations (CSRelation). In MFC terminology a sketch is a document, derived from the MFC CDocument class. The sketch object is responsible for the creation and the deletion of objects as well as relations, its properties include sketch specific parameters, such as sketch annotations, number of objects, current status of the sketch, or the current view. The sketch's functions cover everything that is necessary to create and maintain the two lists as well as global functions for processing its data.

3.2. *Objects*

Each sketch object is a unique entity with its own functionality and its own data space. Originally an object is defined as a set of stokes that has been considered belonging to the same entity (a CSObject object).

3.2.1. Stroke and Point

A stroke (CStroke) is an unintelligent polygon that is created when the virtual ink starts to pour out of the pen and is closed or completed when the user lifts the pen. Such, a stroke stores the geometrical and temporal information of the pen movement (x/y and t). This raw information is captured in a list of points (PPoint structure), which represents the basic data structure of user input within this application.

```
struct PPoint
{
   CPoint  Point;               // Coordinates of input point
   CPTime  Time;                // Time when point was created
   int     PenSize;             // Pressure dependent PenSize
   int     Flag;                // Point specific flag
};
```

Code Fragment 1 Definition of the basic PPoint structure.

Before a stroke can be used it is preprocessed and irregularities, such as duplicate points are being eliminated. The number of points per stroke is further decreased by applying a smoothing function (See also Section 4.1).

3.2.2. An Object

The general type of a sketch object can be `Region`, `Line`, or `Point`. However each objects can maintain a set of regions, lines and/or points. This is because an object can consist of more than just one of these elementary forms, for instance a town can be graphically represented by multiple boxes representing houses. Beside all elements being of the same type it is also possible that an object can have mixed forms, e.g. lines and regions. However, each object can only have one geometric type (region, line, or point) that is assigned when the object is complete. This type is also manually changeable by the user. During the initial drawing phase of the sketch all objects are either regions or lines. Objects of type point can only be created when two line objects intersect that is, points represent always derived objects.—By now this feature of object inference is only conceptually implemented, hence the prototype generates no derived objects. However the principal mechanism is already implemented in the current data model (See Section 6, Future Extensions).

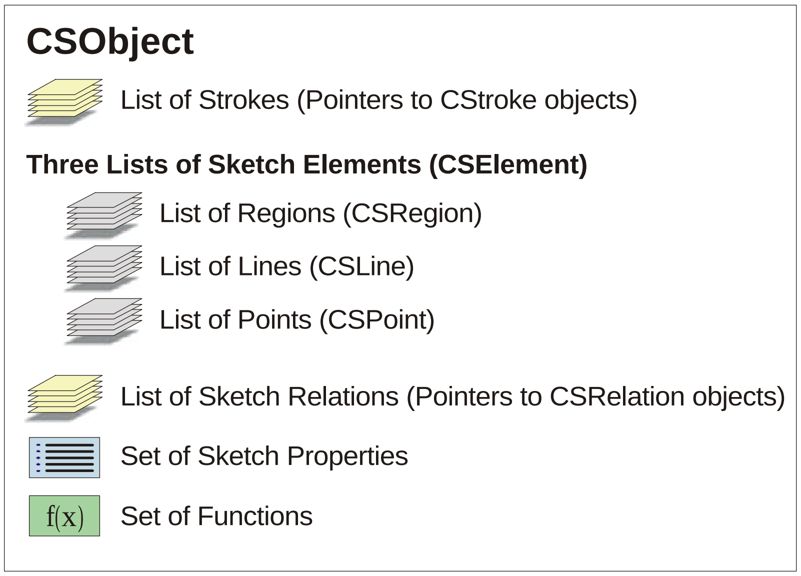

CSObject
List of Strokes (Pointers to CStroke objects)
Three Lists of Sketch Elements (CSElement)
List of Regions (CSRegion)
List of Lines (CSLine)
List of Points (CSPoint)
List of Sketch Relations (Pointers to CSRelation objects)
Set of Sketch Properties
f(x)
Set of Functions

Figure 3 General overview of the CSObject class

The set of properties of an object is relatively big this is, because a sketched object stores all the object relevant data, such as annotations, geometrical parameters, state of object, and many things more. Similarly, an object has a huge functional interface. An object is able to process itself, it can draw itself in different views, or change its type. Summarized, it is a self standing unit with “knowledge” about itself and its close environment. The link to the outside world is provided through a list of pointers to relations that bind the object within the sketch with other neighboring objects. These pointers represent a subset of those that the sketch (CSketchoDoc) maintains.

3.2.3. Sketch Elements

Sketch elements are the “intelligent” form of strokes, that is they have a type (`Region`, `Line`, or `Point`) and they have some higher order functionality. The base class for all sketch elements is CSElement. Derived from this class are CSRegion, CSLine, and CSPoint. The information stored in an element is purely geometrical and no semantic information can be stored. Elements are automatically derived from strokes during the object assessment phase, when strokes are broken down into non-intersecting segments and objects are interpreted. The functionality of an element can vary, depending on its type. Line elements, for instance, have a length while region elements have an area function. An other difference between objects and elements is that an element has no information where it belongs to, this linkage information is only stored within objects.

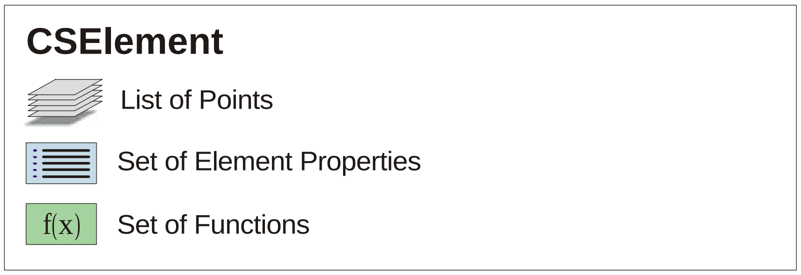


Figure 4 General overview of the CSElement class, which is the super class for regions, lines, and points.

Because objects derived from subclasses of CSElement are purely geometrical, they are also being used for other purposes, such as storing intermediate results or calculating other parameters, such as the convex hull of an object.

3.3. Relations

There are two types of relations supported by our the prototype, binary relations (CSBinRelation) and multiple relations (CSMultiRelation), both are derived from the CSRelation class. Obviously a binary relation links two objects and a multi relation more than two object with each other. Relations are considered as self standing entities within a sketch, similarly to sketched objects. The major difference between sketch objects and relations—beside interface and data space—is that objects are drawn and relations are not. That is, relations are in general not explicitly specified by the user, but derived based on the configuration of the sketch.

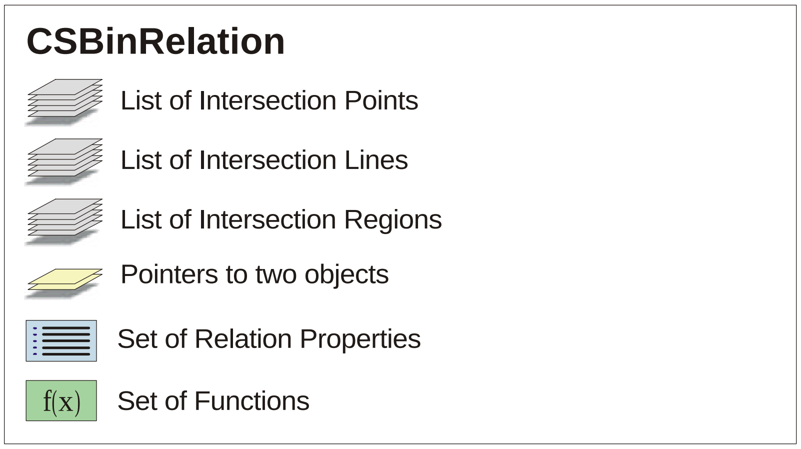

CSBinRelation
List of Intersection Points
List of Intersection Lines
List of Intersection Regions
Pointers to two objects
Set of Relation Properties
f(x)
Set of Functions

Figure 5 General overview of the CSBinRelation class

By now the system creates only binary relations. However, multiple relations can be defined manually between three or more objects, though there is not yet any implemented functionality that goes over the basics of creation and destruction of a multiple relation. Binary relations, on the other hand, are created by the application at the time when all objects are drawn and the sketch is ready to be processed. Once an empty relation between two objects is created, then the relation can derive all the necessary parameters, such as metric, topology, and directions autonomously. To do so, a relation maintains pointers to all objects it is related to. If two objects intersect each other, then the relation is also storing their geometrical intersection, which can be a sets of points, lines, and/or regions. These sets of intersection are by now only used to calculate relational parameters, but they could easily be used to create derived objects in the future.

4. Sketch Processing

During the process of extracting information from simple strokes it is essential to take the spatial and temporal component into account. This approach provides valuable information about the genesis of a sketch, in that it may reveal step by step what the user was thinking while sketching a particular situation (Blaser, 1998).

People typically draw objects one by one, without switching between objects while drawing. Applying such a priori knowledge, it becomes much easier to make the first analysis of a sketch. But still, if the examination would only consider temporal aspects of drawings, then this would be insufficient, because the user might want to add or edit a previously drawn object and such interrupting the normal sequential creation of objects.

The first part of this section is concerned about parsing the input stream of points and about extracting and associating strokes with objects. The second part takes a look at how the prototype processes and classifies raw objects, and the third section, finally, describes how objects are related to each other.

4.1. Sketch Parsing

The sketch parsing part consists mainly of a first quality check combined with a simplification method that reduces the number of points while retaining the original information content and the shape of the stroke. This phase is immediately followed by an first evaluation of the stroke and an attempt to associate the stroke with a previously drawn object.

4.1.1. Smoothening

Depending on the drawing speed, the used computer, the input device, the operating system, and in general the whole sketching environment there is an abundance of points that are passed to an event-based application. Many of these points are redundant or unnecessary, because the point density is much too dense. Other impeding factors are that sometimes points in temporal sequence have identical values and that sketched objects are frequently imprecise and shaky, which is due to the very nature of any freehand drawing.

In order to cope with such an irregular and unpredictable input flow it is an essential part of any such system to be able to filter points. For our prototype we chose the Douglas Simplification Algorithm (Douglas and Peucker, 1973; McMaster and Stuart, 1992) that uses a holistic approach to eliminate all unnecessary points from a line.

The algorithm uses one single tolerance value to determine if points should be kept or dropped. Figure 6 below shows how the algorithm is used for a section of a line segment. The blue fine line is the original line and the red line is the result of the simplification.

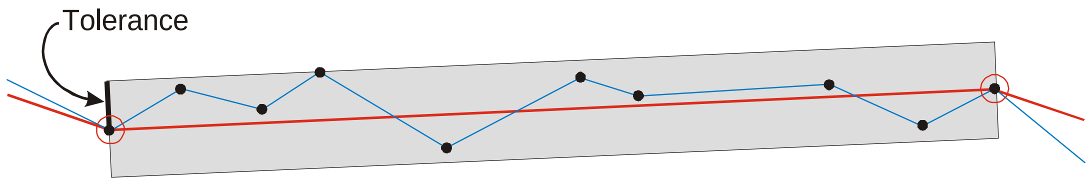


Figure 6 Principle of the Douglas Simplification Algorithm

This algorithm is very robust and efficient, and with a carefully chosen tolerance the difference between the non-smoothened and smoothened stroke is barely visible. The great advantage of reducing the number of points at this stage is that less points translate directly into less segments, what greatly reduces processing time.

4.1.2. Sequencing

The second task is associating strokes to objects. For each newly created stroke there are basically three possible cases, if we exclude the case of the first stroke:

- a stroke can be associated to the current incomplete object,
- a stroke can be associated to an other than the current object, or
- a stroke can be associated to no other previous object that is, it is the first stroke of the new current object.

The criteria for this classification are based on the time difference between the last previously drawn stroke and the new stroke as well as on the location of the new stroke in relation to other objects. In this context we use some user configurable thresholds for each of the applied measures. Figure 7 shows the principle for the time parameter.

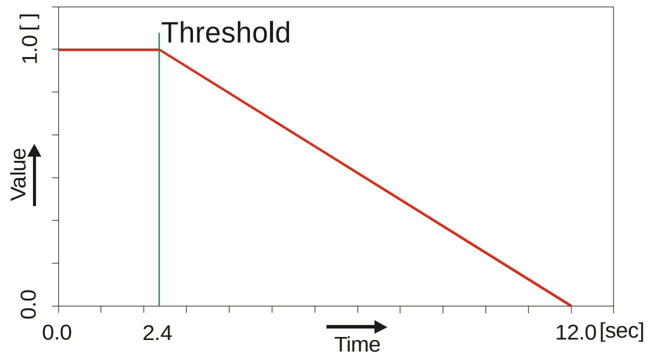


Figure 7 Basic principle of the influence of time for the sequencing of strokes.

In the example in Figure 7 all strokes that are drawn within a timeframe of 2.4 seconds are automatically associated to the previous object. After this period of time the “Connection value” shows a linear decrease. The same function is used for all other measures as well and the values of each function are added and normalized, if no measure indicated a value above the threshold (Formula 4.1). This normalized value must be greater than another user definable threshold, if the stroke is to be considered belonging to the previous object.

$$\text{Connection Criterion} = \frac{1}{n} \cdot \sum_{i=1}^{n} \mathit{ConnectionValue}_i \qquad \text{Formula 4.1}$$

The reason for this approach is that people have different sketching strategies that must be taken into consideration. The prototype does no reasoning about these parameters, the preferences must be configured manually using sliders. However, a future application could analyze the number of sequencing errors and accordingly adapt certain parameters. Such user preferences could be stored and used much like the profiles used for speech recognition.

The application informs the user about the sequencing of strokes by coloring objects differently, as described in Section 2. Strokes that have been associated with the wrong object can easily be re-associated by applying the attach or detach button. This action can also be done at later time when the sketch is already finished. Below is a list of editing tools that the prototype provides:

- Attach or detach a stroke or groups of strokes to or from an object
- Break objects apart into single strokes (ungroup)
- Group strokes or objects to an object

- Move or delete strokes or objects.

According to our survey these functions where most important to the surveyed subjects, when asked about essential functions in a sketch-based query interface for GIS.

4.2. Object Generation

When a set of strokes has been grouped to an object and when it has been closed—that is, the next object has been created—, then it is time to analyze and further classify this object. The presented measures represent a basic set of functions that can be extended in a later phase of the development of a sketch-based application. Knowledge about objects at this stage is crucial, because it allows the system to make a pre-analysis of the sketch, solely based on detected objects and without yet taking their relationship into account. For instance, it is feasible that a software agent starts to browse remote databases in order to detect objects and related information just after a sketched object was analyzed and *before* the actual query is formulated. Such, information and meta information is already available when the user presses the query button, essentially speeding up processing and the search for information. On the other hand this pre-processing allows the system also to interact with the user and to advise him or her about certain pre-fetched facts (Blaser, Sester and Egenhofer, 1998).

The following sections describe some basic object classification measures.

4.2.1. Text Detection

A user of a sketch-based system can annotate object by three different ways, he or she can type, write (freehand writing), or talk to the system. While the talking aspect is conceptually prepared it is not yet implemented and so is the handwriting part. However, the present prototype can distinguish handwriting from ordinary objects and such, making it easy to process an object that has been detected as text.

Text detection in this phase is necessary because written text is different from ordinary objects in that the point density of test strokes must be much higher to allow for a good handwriting detection. Therefore, the initial smoothening described in Section 4.1.1. must be done with a conservative tolerance. This is the reason why we smoothen objects that have been classified as non-text objects a second time—using a higher tolerance—right after the text detection.

The criteria for the text detection are based on observations described in our sketch report. For instance, we noticed that most people keep a certain writing angle, writing from left to right. Beside this, we consider the curvature, the number of abrupt direction changes, and the overall direction change of strokes to decide if a group of strokes is to be considered text or a sketch object.

4.2.2. Gesture Extraction

Gestures are something very natural to human beings. They are articulated using a variety of “tools”, such as hands, the face, and even the whole body is frequently used to “talk” in this non verbal language. In sketches gestures are reoccurring symbolic drawings that are borrowed from various disciplines. Although not standardized, we found that there is a common set of gestures that is used and understood by a large number of people from the same cultural environment. An good example is crossing something out. Other gestures are more domain specific, hence they often represent common symbols of such a domain; a sketch triangulation point in a surveyor’s sketch is an example ◬ .

Gesture detection and interpretation is very useful because it provided an other, alternative set of communication tools that is relatively swift and lean compared to other approaches in user computer interaction. A sketched gesture for instance, such as a bridge symbol is drawn quickly, with two simple strokes but it conveys a lot of information.

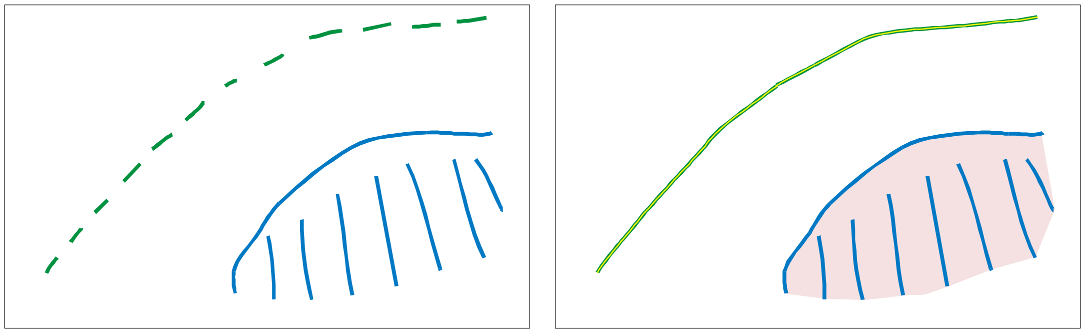


Figure 8 Example for dashed line and hatch detection; left the sketch and on the right side the interpreted result.

The prototype detects two types of gestures: dashed lines, and hatched areas, as shown in Figure 8. Other, simpler structures that do not have a varying number of strokes, such as a cross (e.g. with two strokes) are much easier to detect, hence is seems feasible to incorporate an entire, extensible and user configurable vocabulary of gesture for a sketch based application.

4.2.3. Segmentation

The segmentation of objects is a relatively difficult task this is, because of the virtually unlimited possibilities people can create their sketches. The term segmentation in this context means that all strokes are broken down into non-intersecting, connected segments, such as shown in Figure 9.

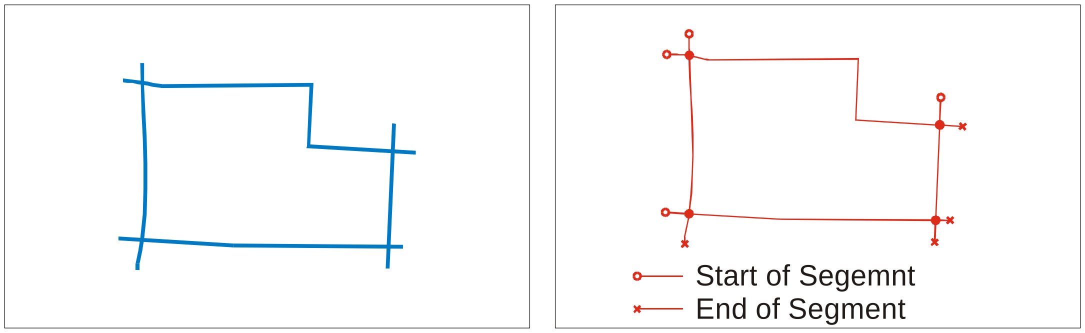


Figure 9 Segmentation of an object: Left before the segmentation (four strokes) right after the segmentation of the object (twelve segments).

Each segment has a list of points, a start and an end point and some “knowledge” about the immediate neighbors in form of two other lists that store pointers to these neighbor segments and some other related information. A segment can be closed, self-closed, half-open, or it can be completely open. This information is crucial for all subsequent processes that rely on how segments are connected with each other.

4.2.4. Object Completion / Object Clean-Up

Sketching is no exact science and people often sketch their objects incompletely and inaccurately. This leads to gaps and irregularities in objects that are difficult to process if they are not previously corrected. People, on the other hand, have no problems recognizing even very incomplete figures, because they can anticipate the correct shape or silhouette solely based on a few fragmentary strokes as is shown in Figure 10 below.

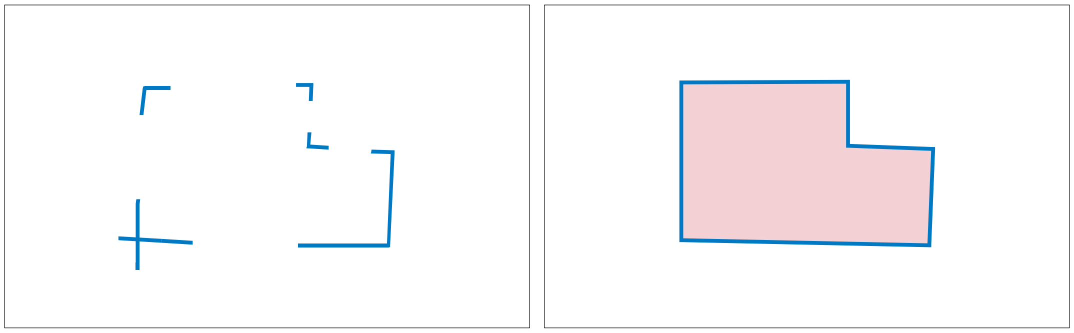


Figure 10 This Figure shows a fragmentary sketched object on the left side and the object that most people would perceive on the right side.

Although the example in Figure10 is somewhat exaggerated, gaps and incomplete objects are very common. In most cases strokes do not exactly intersect where they are meant to and such they stop before they can intersect another stroke or "overshoot". The prototype covers three of the most common cases, to correct these problems (Figure 11).

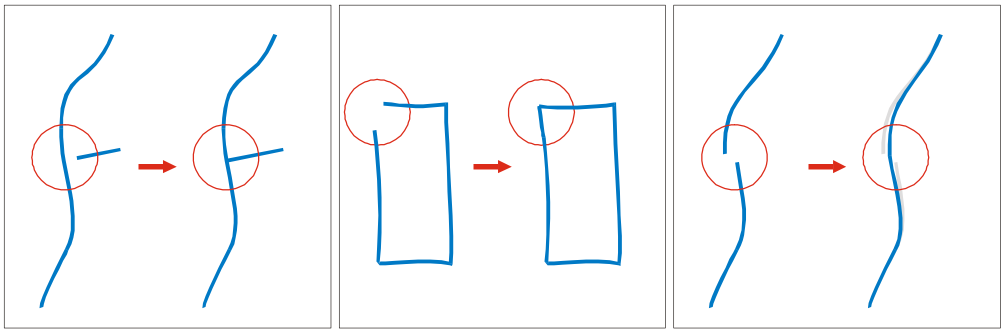


Figure 11 Three cases where object completion is necessary.

Object completion is done in multiple steps with different threshold levels, similar to the process of smoothening strokes. The first time that a stroke is analyzed in this regard is directly after it has been associated with an object. At this point we check if the stroke must "snap" to the object. For this purpose there are again different user adjustable, snap-distances for strokes that end close to a vertex of another stroke and those that end close to an edge. The second phase of object completion is scheduled after the object has been segmented.

Object clean-up is simply the process of dropping segments that are very short and partially connected or fully open. This is often the case when two strokes intersect and one or both resulting segments.

4.2.5. Outline Extraction

The next step is to extract outlines. For this purpose the application tries to find the largest closed area of the object. If there are more than one area then they are all stored in a sorted list, depending their size. If there is no closed area then the convex hull is calculated and stored as a quasi area. The prototype supports multiple areas that can contain each other. The only restriction is that closed regions may not touch, in which case solely one region (the larger one) is stored.

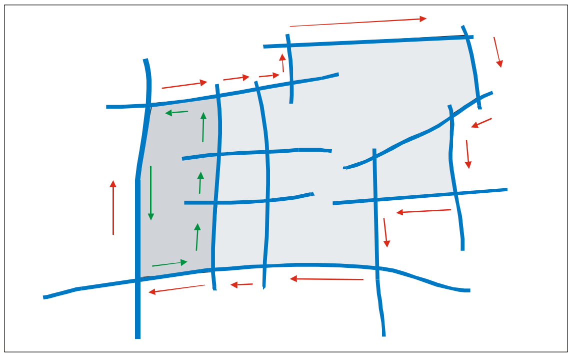


Figure 12 The two areas resulting from different turning directions.

The principle of outline extraction is to follow segments and to turn always to towards the same side at intersections, where more than two segments come together. This procedure has to be done at least twice for each object, because it is very difficult to predict the appropriate turning direction a priori. The results are two areas that may differ in their size (Figure 12). The bigger size is the outline. This procedure is repeated until all segments are processed, that is until there are no closed segments left that have not yet been processed.

4.2.6. Type Determination

The type of an object is determined by comparing the sum of areas of all region of the object with the area of the convex hull of the object. If the sum is smaller than a certain percentage, then the object is considered a line object—even if there are regions in the object. Using this approach, we can also have mixed type objects that is, objects that contain more than just one element type. If the system should misinterpret an object, then the user can change the type of an object by the push of a button.

4.2.7. Parallel Extraction

If an object has been considered being a line, then the application tries to find parallels within the object. For this purpose we check for every segment if there are neighboring segments that do not belong to the same stroke but that are in reach and parallel to this one. If parallels are found we check if there are other parallels within the same object that are potential meet candidates. Figure 13 shows an example of a road intersection where such a procedure is useful.

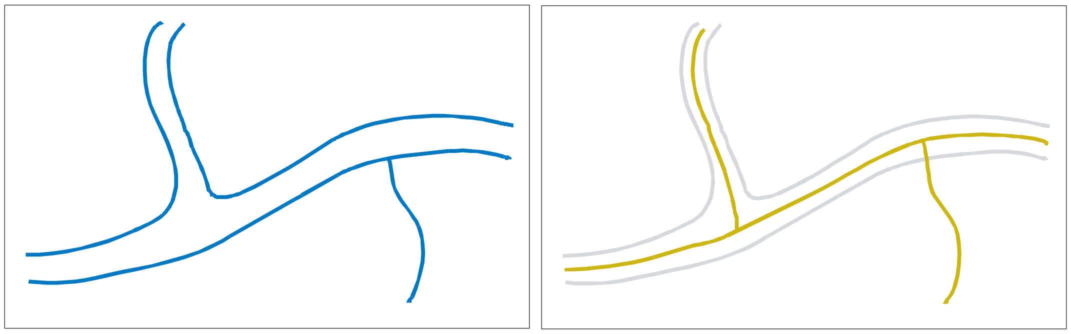


Figure 13 Left a line object before the parallel extraction and on the right after the parallels have been detected.

Detected parallels substitute all segments that have been involved in their extraction. These parallels and the remaining segments are stored in a list of lines within the object (Figure 13). This list of lines is sorted depending on the length of each line.

4.2.8. Kernel Extraction

We define the kernel of a region as the longer centerline of the tilted minimum bounding rectangle (TMBR) of an object. This is an abstraction for the real centerline of an object that is more complex to calculate. The TMBR is that bounding rectangle that has the smallest possible area of the whole set of bounding rectangles for a specific object. Such a bounding rectangle is obtained—with an adequate accuracy—by rotating an object with small increments (1.0° grad in our case) and recording size and position of the MBR's. The advantage of a TMBR is that it is only slightly more complex than a regular MBR but has, compared with this one a directional component and an optimal fit for a specific object. Furthermore, most objects in sketches are boxes anyway (Blaser, 1998), and such a TMBR is an ideal approximation for many calculations.

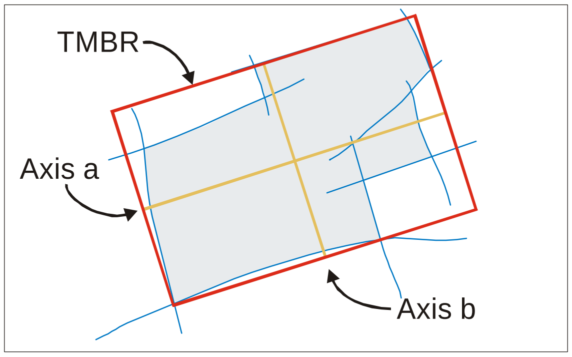


Figure 14 An object of type region and its TMBR with the two main axis a and b.

Figure 14 shows a TMBR and the two centerline axis (a and b) of a region object. TMBR's can also be used for the approximation of lines, but due to their specialties a TMBR is less suitable for this purpose. However, TMBR's carry in any case—for lines and regions—more information and a more accurate than their MBR counterpart. A TMBR can be defined in different ways, depending on external constraints, such as processing speed or compactness. In our case we have opted for speed and stored all four corner points.

4.3. Relation Extraction

The subsequent step after the sketch has been parsed and all objects have been assessed is to bring these objects into a meaningful relation to each other and to describe these relationships in a formal way. The following sections describe the foundation and realization of the approach chosen in our prototype.

4.3.1. Relational Network Generation

A common approach to establish binary relationships between objects is to connect every object with every other object. The actual number of binary relations for n objects applying this method is: `n!/2*(n-2)!`. While this method has the advantage to be comprehensive in that no single relation gets lost, but it has the serious downside that many of these relations are unnecessary, because there is only a virtual relation but no real connection between two objects. An example is a suburban district with 1000 houses, the likelihood that one house of the east side has a direct connection to one house of the west side is relatively low. Hence it seems to be beneficial to establish and calculate only a selection of relevant relations.

During the implementation phase of the prototype we have been more and more aware about the necessity of developing a theory and a method that can intelligently and efficiently distinguish between important and irrelevant relations. We are currently investigating approaches that can be used for sketched objects including points, lines, and regions. Because there is yet no theory available we have implemented a simple algorithm that is based on geometry alone. The basic idea of this method is that every object has a certain range ("relational range") that is dependent on its and its neighbor's objects size. The size in our case is the area for region objects and a function of the length for line objects (Value for the "area" of a line:= $\mathtt{Length}^2\mathtt{/4}$, with 4 being an arbitrary, user definable value).

$$\text{Range of Object} = \sqrt{Area_{ObjectA} + Area_{ObjectB}} \qquad \text{Formula 4.2}$$

This approach provides in many cases a good approximation that can later easily be substituted by an other, more elaborate algorithm.

Each relation is considered and treated as special sketch object with no drawn features but instead with links to such draw sketch objects. Relations have their own data space and their own functionality. Therefore, it is straightforward to extend the interface of such a relation when new findings are being made. The following three sections reflect the current status of our research concerning binary spatial relations.

4.3.2. Topology

This research area is relatively well documented for simple regions, simple lines, and points (Egenhofer, 1989; Egenhofer and Franzosa, 1991; Egenhofer and Mark, 1995). The prototype implementation covers all *Region¾Region* (RR) relations and a subset of *Region¾Line* (LR) relations.

Because sketching is no exact science we have adapted the theory in so far that we have introduced a tolerance for certain topological RR conditions, such as Meet, Covers, and CoverdBy. The motivation for such a tolerance is that we primarily want to assess what the user has intended to draw and only secondarily what he or she has actually sketched. Figure 15 shows an example of a situation where a user placed two objects such that they touch (meet condition). With a pure mathematical approach, however this would be a clear overlap condition.

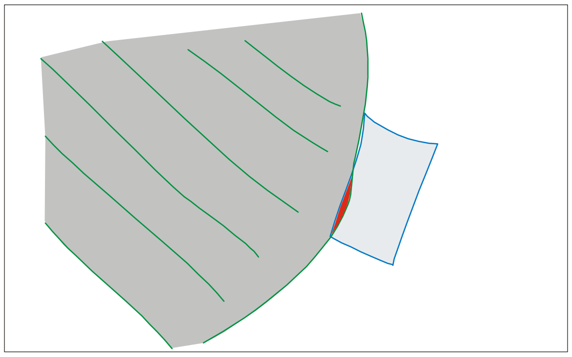


Figure 15 Binary RR object relation, that is considered to be of type Meet, despite the small section that the two objects overlap.

A tolerance must be introduced with all other spatial relations that have similar characteristics, such as when a road object (Line) is sketched following the boundary of a settlement (Region). Because people have different sketching strategies these tolerances, like most other adjustable parameters, have to be stored in a user profile.

4.3.3. Direction

The directional relation between two objects is described by the Direction-Relation Matrix (Goyal and Egenhofer, 1997). This is a three by three matrix that describes “how much” neighboring object is in each of the eight adjacent, bounding boxes and in the minimum bounding rectangle of the object itself. These values are normalized, such that the sum of all values is 1.0. Figure 16 shows this definition graphically .

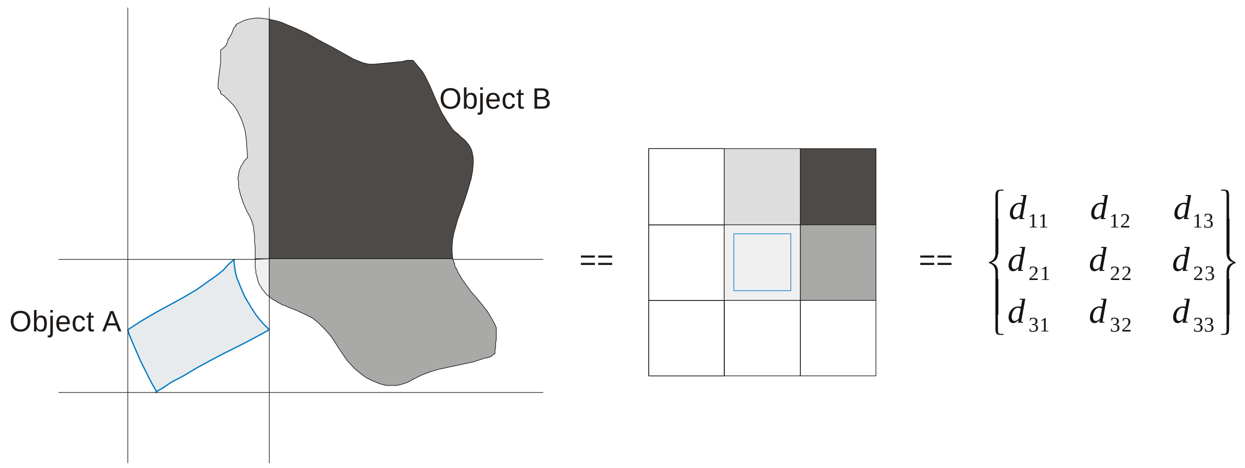
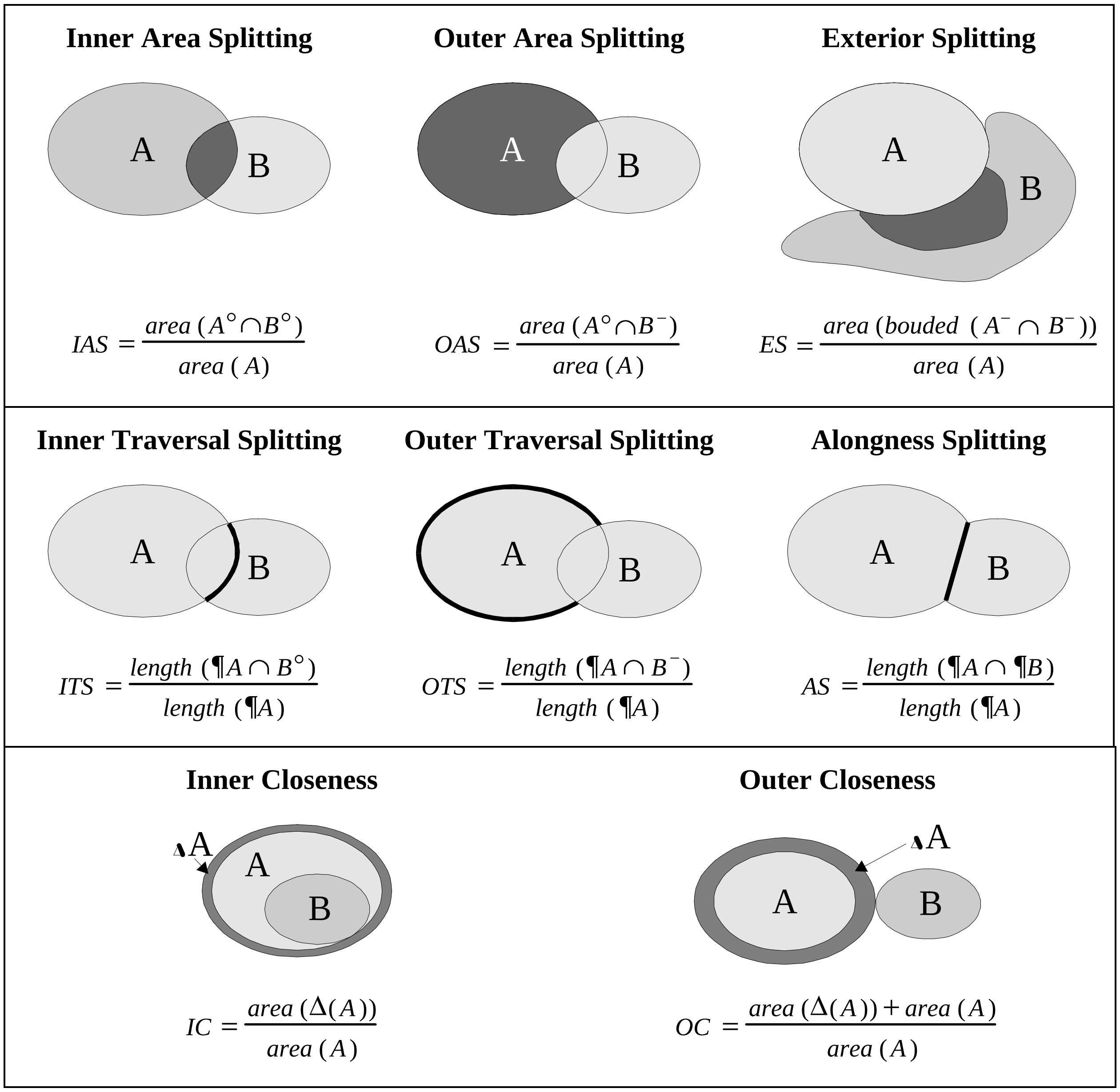

Object B
Object A
==
==
$\begin{Bmatrix} d_{11} & d_{12} & d_{13} \\ d_{21} & d_{22} & d_{23} \\ d_{31} & d_{32} & d_{33} \end{Bmatrix}$

Figure 16 Direction-Relation Matrix for an object A with an object B in graphical and numerical form.

The values of the Direction-Relation Matrix are derived from the percentage of the object in each of the nine quadrants, this is an area for region objects and a length for line objects.

4.3.4. Metric

For each set of binary topological relations that is possible between two objects there are formal, metrical descriptors that circumscribe this relation. The formulations used in our prototype application are primarily based on research done by Shariff and Egenhofer (Shariff, 1996; Egenhofer and Shariff, 1998), who describe eight metrical refinements for Region—Region relations (Figure 17).



$$IAS = \frac{area\ (A^{\circ} \cap B^{\circ})}{area\ (A)} \qquad OAS = \frac{area\ (A^{\circ} \cap B^{-})}{area\ (A)} \qquad ES = \frac{area\ (bouded\ \ (A^{-} \cap B^{-}))}{area\ (A)}$$

$$ITS = \frac{length\ (\P A \cap B^{\circ})}{length\ (\P A)} \qquad OTS = \frac{length\ (\P A \cap B^{-})}{length\ (\P A)} \qquad AS = \frac{length\ (\P A \cap \P B)}{length\ (\P A)}$$

$$IC = \frac{area\ (\Delta(A))}{area\ (A)} \qquad OC = \frac{area\ (\Delta(A)) + area\ (A)}{area\ (A)}$$

Figure 17 Metrical refinements of topological relations between regions

Similar refinements are also possible for Line—Region relations, but they are not yet implemented in our prototype.

5. Levels of Abstraction

Our prototype has three different levels of abstraction for the same data. In Microsoft MFC terminology different levels of abstraction are called *Views*. The primary data view is the Sketch View, this is the place where the sketch is generated and edited and this is also the only mandatory view. The other two views can be used to examine the intermediate results of the sketch interpretation.

5.1. Sketch View

This is the normal environment of the user interface, it is the place where the sketch is initially created, edited, and from where a spatial query can be made. The user Interface is very simple, including only few primary drawing tools (draw, select, zoom, delete) and even somebody with only limited computer experience can readily draw a simple sketch—It worked fine with a four and six year old child—. If the mouse as input device is substituted with an electronic pen and used in conjunction with a tablet or directly on a flat screen, then the usability and simplicity is further enhanced. In our experiments we have used a Wacom 12x12" drawing tablet and a pen with a double switch button (rocker switch). One button is configured to invoke extended functions of the pen and the other to get context information or set object properties, depending on the actual cursor position.

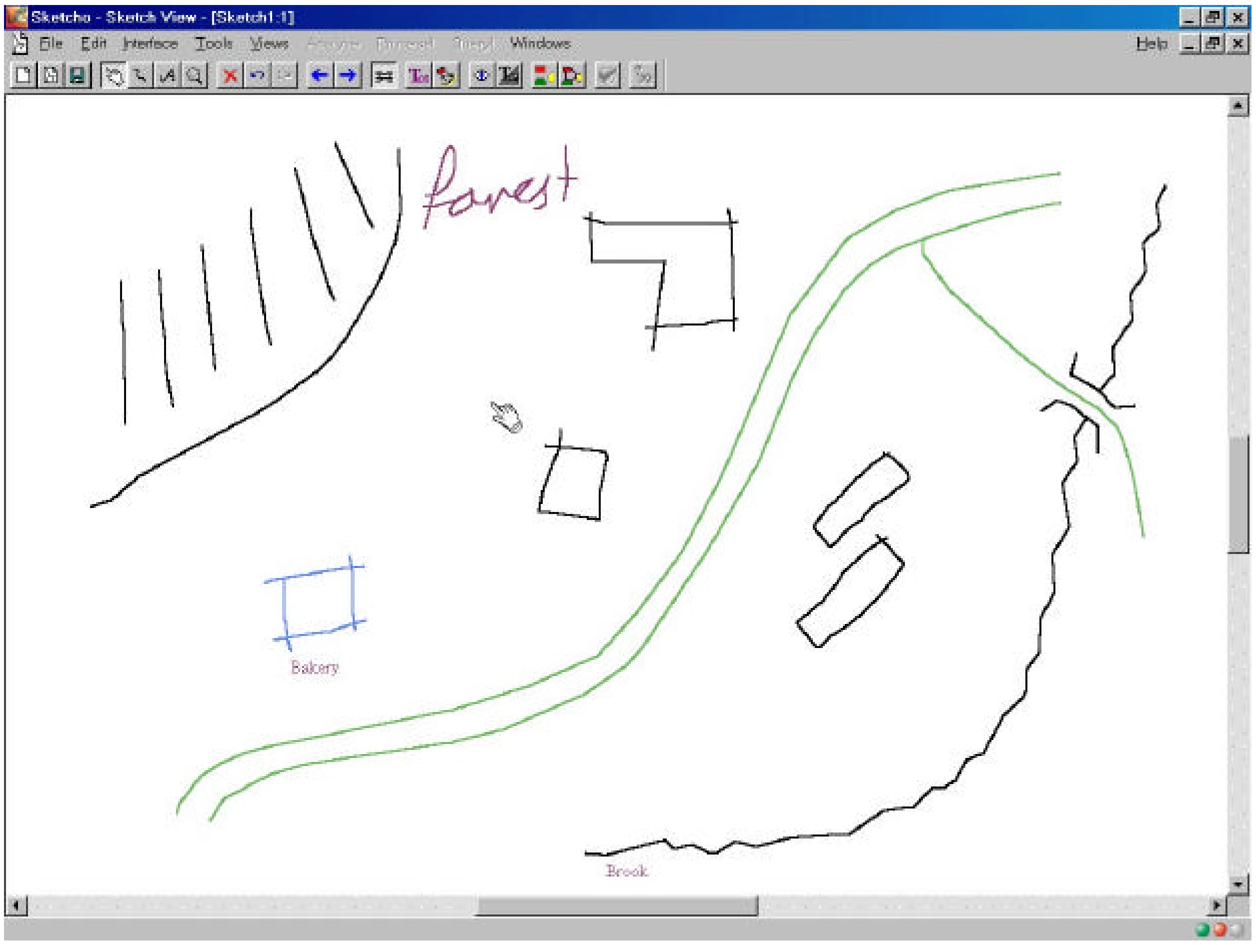


Figure 18 Sketch View with a typical sketch.

The cursor changes its appearance and takes on different icons, depending on the current mode, so that the user is always aware about the present status of the pointing device. Other visual clues, such as different object colors, have already been mentioned in the user interface

Section 2. Figure 18 shows a screenshot with a typical sketch. An extensive description of the functionality of this initial user interface can be found within the users guide to the prototype application (Blaser, 1999).

5.2. Object View

The Object View shows what the application has detected in point of view objects. The view shows a simplified and interpreted outline of each object and it conveys—using different colors—each object's type. Should an object have been misinterpreted, for instance a region has been considered a line because it was not automatically closed, then this can be reversed by a simple push of a button. It is also possible to move, delete, and edit objects at this stage, all without changing the initial sketch. That is, modifications on this level are only passed forwards to the next level of abstraction, but not backwards.

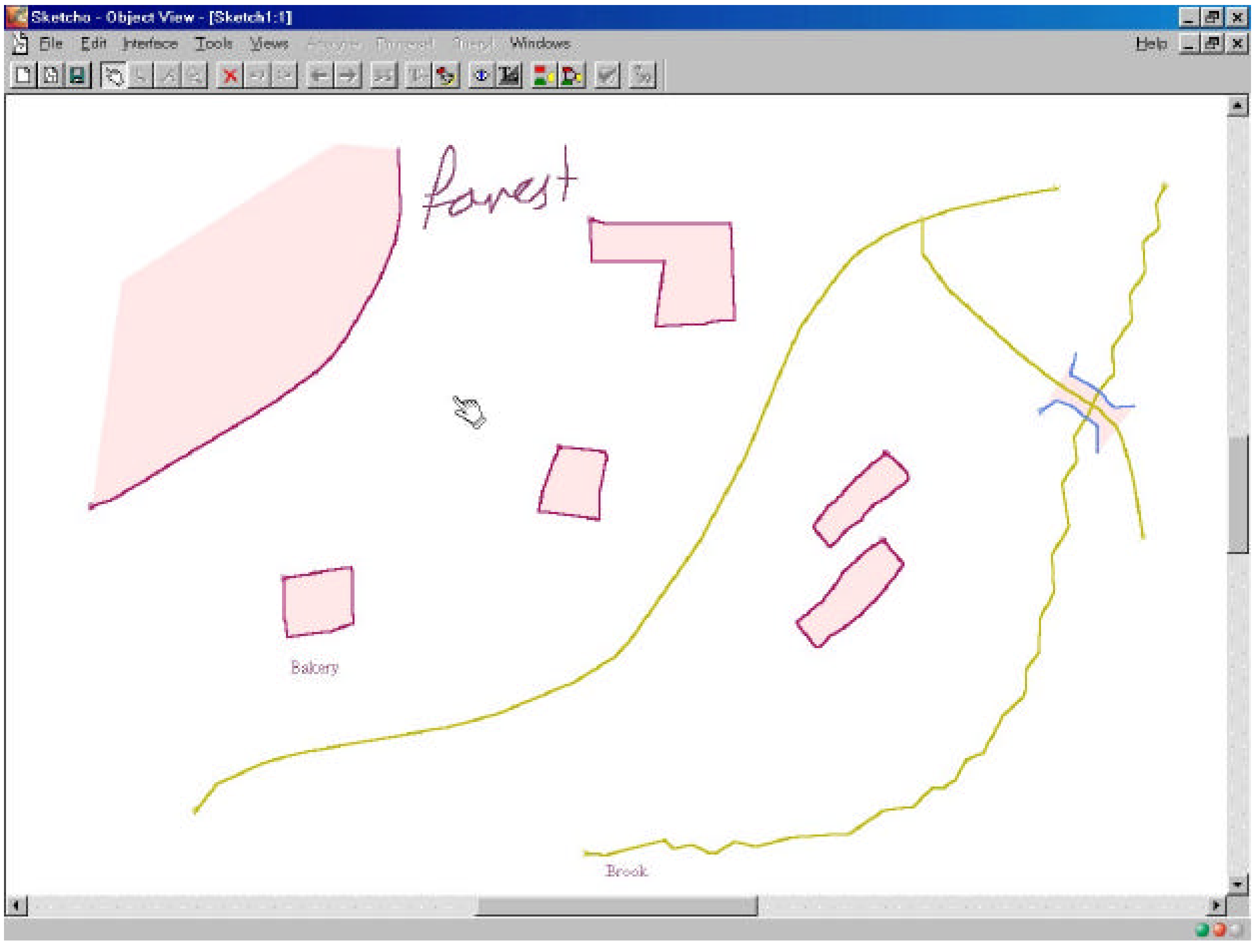


Figure 19 Object View showing the same situation as in the previous section

This view is well suited to examine and control the automated processes of the prototype. In the example of Figure 19 all objects have been detected correctly, solely the bridge symbol on the right side of the sketch was improperly classified as line object. Upon detecting such an inconsistency, the user has quickly converted the bridge into a region, by pressing a single button. On the other hand all regions have been correctly closed and the forest was detected as an open (hatched) region. Considering line objects, the processing was done satisfactory as well that is, the parallel lines have been substituted by a single line, the adjacent road has been attached, and the two interrupted brook segments have been connected as well.

5.3. *Formal View*

The Formal View (Figure 20) is the third and last view of our prototype application. This view represents the highest level of abstraction where every object or relation is solely depict by a symbol that is connected to other entities according the previous sketch analysis. Currently there are five different symbols implemented. These symbols cover the present data model of the prototype, which are: region and line objects, and binary and multiple relations. As mentioned before, multiple relations can be established, but they do net yet carry any data, but that necessary for the connection of the selected objects.

Despite the relative simple and abstract look of this view, the user has some powerful tools to further edit the sketch in this phase. For instance, it is possible to disable or re-enable relations or objects, and it is as well possible to create new relations between objects even if there is no previous automatically generated relation.

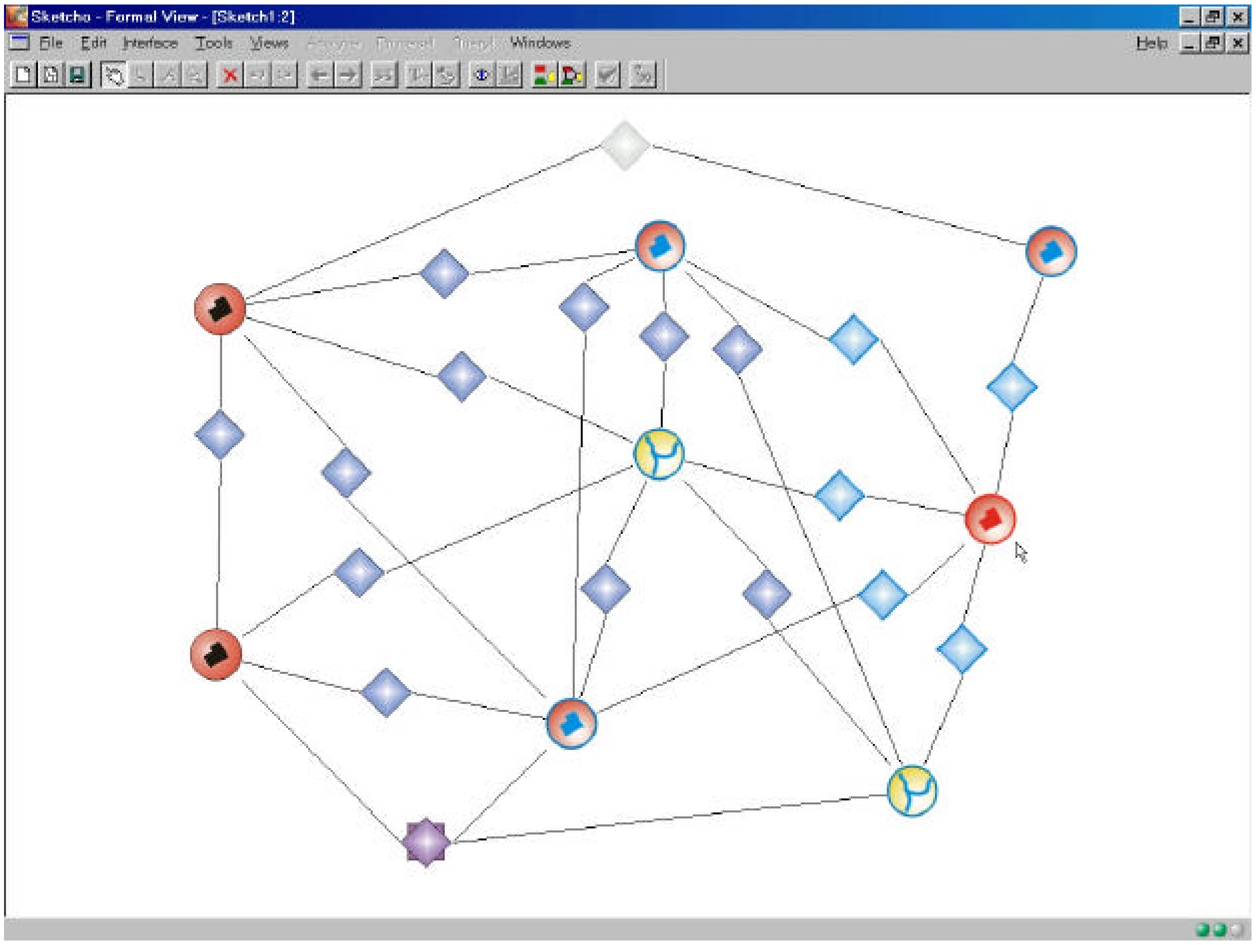


Figure 20 Formal View showing the same situation as in the previous section

It is also possible to delete object and relations permanently, but like in the previous view, such an action has no effect on the original sketch or its representation in the Object View. An other feature of the Formal View is the browsing component: Properties of objects as well as those of relations can be viewed and partially modified. However, modifications are limited to realistic parameters, such as the object relevance or topology.

6. Future Extensions

Like with many other research projects, this project has produced more questions in certain areas than answers. This is not necessarily negative, first because this is the very nature of research and second, because some questions, problems, or ideas can only evolve if certain paths are investigated. The following sections are a composition of thoughts and ideas that emerged during our research and the subsequent implementation of our prototype. Some ideas have already been conceptually or partially implemented while other topics are new and not yet put on paper.

a) Integration of Verbal Input

The integration of a verbal component into our sketch-based prototype has been a long term goal and with recent advances made in speech recognition this goal seems much more feasible than ever. The complexity that such an integration would add is—from our point of view of the application—mostly due to the synchronization between spoken word and the sketched objects. The system has to conceptually and semantically "understand" what the user is drawing and check the consistency of this multi-modal input. For instance if the user draws a box and talks of a road, then the system has to "know" that this verbal annotation is most likely not meant for the box, but maybe for the previous or following, not yet drawn, object. That is, the application has to keep track of objects that have already been created and "keep in mind" that the user might speak of objects that are solely created in his or her mental model of the sketch but not yet on paper. Such an evaluation and consistency checking process requires a great deal of implemented intelligence and a solid procedural model. A research group at the NCGIA in Orono has recently started to examine possible approaches in this context (*Sketch and Talk*).

b) Handwriting recognition

A true multi-modal application has to be able to understand handwritten input as well as spoken verbal input. However, the integration of this functionality seems to be a much easier task, because the current prototype is already capable of distinguishing sketched objects from handwritten text. Such, the actual task is reduced to examining the detected text strokes for characters and known words.

However, and similar to spoken input the processing does not stop here, because a textual annotation must be associated to an object so that the information can be used for the interpretation of the sketch (Blaser, 1998).

c) Gesture Detection

Gestures are similar to symbols on a map reoccurring signs with a specific meaning. Our prototype can detect two more complex gestures (irregular dashed line and hatched open area) but there are many other common gestures that a sketch-based application should be able to understand. Hence, the task in this concern is to compile databases with domain specific and domain independent gestures and to develop a mechanism that allows to personalize and use these gestures within a sketch-based application. Because of the similarity to other well-researched fields, such as handwritten character recognition, this task seems to be practicable as well.

d) Personalized preferences

Because sketching is so individual from one user to another, it appears that personalized settings must be automatically generated for each individual user. A sleek method could consist of a mechanism that traces a user's actions and that would learn a user's preferences "on the job" that is, for instance, when the system makes a misinterpretation and when the user has to manually corrects this error, then the system should evaluate this correction and adapt the preferences accordingly. For instance, if a user constantly has to attach strokes to the previous object, then the system should extend the timing parameter in the parsing phase.

e) Theory for the inference of secondary objects

Primary objects are objects that are drawn directly onto the drawing device, while virtual or secondary objects are in general created implicitly. An intersection of two roads is a good example: although not drawn as such—the location of a simple line-line intersection must not be especially marked—an intersection may play a much more important role than the two line objects it was originally derived from. Hence, we have to extend our formalism to include derived objects as well. For this purpose we need a mechanism that evaluates object relations and that, if necessary and applicable, creates new objects. For each such new object, we have to eliminate one relation and create at least two new relations.

f) Theory for the generation of a network of relevant object relations in a sketch

In Section 4.3.1 we have mentioned that it seems unnecessary to take all possible relations in a sketch into account. Rather it appears that it is beneficial to introduce object neighborhoods, where every object has only "knowledge" about its immediate neighbors. Such an approach leads to a network of connected objects with a substantially reduced set of relations.

Beside the advantage that such a system can be processed at a fraction of the expense that is necessary to calculate a model that incorporates all possible relations, there are other advantages as well. For example, updating and modifying such a reduced system appears to be very elegant, because, instead of recalculating the entire model it is only necessary to reevaluate a small portion of the network, that is where an object has been inserted, deleted, or where an object was altered.

In this context various approaches appear to be practicable: A purely geometrical method could be based, for instance, on Voronoi (Ven-) Diagrams (Voronoi, 1908; Okabe, Boots and Sugihara, 1994) to obtain the local neighborhood of each object. Other approaches could combine semantic and geometric information, leading to more sophisticated, but also more complex relational networks.

g) Aggregation and Hierarchy

Being able to aggregate certain groups of objects could further simplify a sketch and make a spatial query more successful. For instance, if a sketch consists of multiple houses that are all located around a river next to a forest, then this situation could be simplified by aggregating the houses into the semantically equal term *settlement*, and such reducing the number of objects to three. This approach leads to an object-oriented hierarchy.

h) Methods to evaluate the relevance of drawn objects

An other form of refining the quality of a sketch is to pre-evaluate the importance or relevance of sketched objects and their relations. Based on our observations in the sketching survey (Blaser, 1998), it seems that not all sketched objects have the same weight. Hence, if it is possible to automatically rank objects depending on their importance, then this would be a great help, later on, when a query must be formulated. The results of such an evaluation provide also substantial insight into the user's mental model of the sketch, which is in turn essential to derive the purpose of the sketch.

i) Translation of the formal model into processable query statement

The translation of the formal model of the sketched representation into a form that can be processed against a spatial data base is another step that requires some thoughts. The formulation of the query depends, naturally, to a great extent on the set of spatial descriptors that are available in the specific query dialect into that the formal model has to be translated. Hence, it makes no sense that a sophisticated spatial description is translated into a query language that does not support any spatial terms at all.

j) Developing a theory that allows relaxing a sketch

Relaxing constraints within a sketched representation is necessary when a spatial query produced only unsuitable results. In such a case the weights of low priority objects and relations have to be decreased, and certain entities may even be excluded so that the next search is more successful. This may sound simple in principle, but the decision which are the parameters to be modified is not trivial.

7. Conclusion

Our prototype application has proven that a sketch-based query user interface is a viable way to describe spatial situations and potentially also to query spatial data. We could successfully parse and sequence sketched input and effectively extract objects. The classification of objects and the formal description of spatial relations between these objects appeared also to be a solid foundation for the generation of comprehensive scene description that can be translated into a processable spatial data base query statement. We have also shown that all major operation in this context can be accomplished in real time and that the level of today’s computer performance is adequate to support a modest sketch-based system.

The prototype also demonstrates that the complexity of a sketch-based system can be hidden behind an easy to use and intuitive user interface—to many uninvolved first time users, the prototype appeared like a simple drawing program—and that many interpretative tasks can be automated.

However, until such a sketch-based query user interface can be put to good use, there are still some challenging issues to solve, such as how to translate the formal model of the sketch into a spatial query statement, or how to derive additional higher order semantics from a sketch (meaning of objects and sketch). But despite these challenges, we believe that sketching is a powerful tool to express spatial constraints and—based on the concepts demonstrated in our prototype—has its legitimacy as one form of input generation within a rich, multi-modal environment.

8. References

Blaser, A. (1996) Spatial-Query-by-Sketch, User Interface Mockup. Orono, NCGIA.

Blaser, A. (1997) User Interaction in a Sketch-Based GIS User Interface. International Conference COSIT '97 (Poster Session), Laurel Highlands, PA, Springer-Verlag, Berlin.

Blaser, A. (1998) Geo-Spatial Sketches. (Second Geo-Spatial Sketches). Orono, National Center of Geographic Information and Analysis, University of Maine, Orono, Technical Report, 133 pp.

Blaser, A. (1999) Sketcho! Users Guide. (Second Sketcho! Users Guide). Orono, University of Maine, NCGIA, pp.

Blaser, A., Sester, M. and Egenhofer, M. (1998) Visualization in an early Stage of the Problem Solving Process in GIS in Computer & Geosciences (to appear)(Special Issue "Geoscientific Visualization"), pp.

Douglas, D. and Peucker, T. (1973) Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature in Canadian Cartographer 10(2), 112-122 pp.

Egenhofer, M. (1989) A Formal Definition of Binary Topological Relationships. In Third International Conference on Foundations of Data Organization and Algorithms (FODO), Paris, France, Eds. W. L. a. H.-J. Schek, Springer-Verlag, New York, NY, 457-472 pp.

Egenhofer, M. (1996) Multi-Modal Spatial Querying. Seventh International Symposium on Spatial Data Handling, Delft, The Netherlands,

Egenhofer, M. (1996) Spatial-Query-by-Sketch. In VL '96: IEEE Symposium on Visual Languages, Eds. M. Burnett and W. Citrin, IEEE Computer Society, Boulder, CO, 60-67 pp.

Egenhofer, M. (1997) Query Processing in Spatial-Query-by-Sketch in Journal of Visual Languages and Computing 8(4), 403-424 pp.

Egenhofer, M. and Franzosa, R. (1991) Point-Set Topological Spatial Relations in International Journal of Geographical Information Systems 5(2), 161-174 pp.

Egenhofer, M. and Mark, D. (1995) Modeling Conceptual Neighbourhoods of Topological Line-Region Relations in International Journal of Geographical Information Systems 9(5), 555-565 pp.

Egenhofer, M. and Shariff, A. R. (1998) Metric Details for Natural-Language Spatial Relations in ACM Transactions on Information Systems 16(4), 295-321 pp.

Goyal, R. and Egenhofer, M. ((in press)) Cardinal Directions between Extended Spatial Objects in IEEE Transactions on Knowledge and Data Engineering pp.

Goyal, R. K. and Egenhofer, M. J. (1997) The Direction-Relation Matrix: A Representation for Direction Relations between Extended Spatial Objects. Annual assembly and the summer retreat of University Consortium for Geographic Information Science, Bar Harbor Maine,

McMaster, R. and Stuart, S., K. (1992) Generalization in Digital Cartography, Association of American Geographers, Washington D.C., 134 pp.

Microsoft (1999) Microsoft Development Studio, Visual C++ 6.0. Seattle, Washington, Microsoft.

Okabe, A., Boots, B. and Sugihara, K. (1994) Nearest Neighbourhood Operations with Generalized Voronoi Diagrams: A Review in International Journal of Geographical Information Systems 8(1), 43-71 pp.

Shariff, R. (1996) Natural Language Spatial Relations: Metric Refinements of Topological Properties. Ph.D. Thesis, University of Maine, Orono, Maine, pp.

Voronoi, G. (1908) Nouvelles applications des paramètres continus à la théorie des formes quadratiques in Journal für angewandte Mathematik 134,198-287 pp.