Algorithm Visualization System for Teaching Spatial Data Algorithms

Jussi Nikander, Juha Helminen and Ari Korhonen
{jtn, jhhelmi2, archie}@cs.hut.fi
Department of Computer Science and Engineering
Helsinki University of Technology

August 5, 2009

Abstract

TRAKLA2 is a web-based learning environment for data structures and algorithms. The system delivers automatically assessed algorithm simulation exercises that are solved using a graphical user interface.

In this work, we introduce a spatial data algorithm extension to the TRAKLA2 system. It is based on using several different types of visualizations on various levels of abstraction simultaneously. The visualizations show both how data elements relate to each other in terms of spatial attributes and how they are stored in data structures.

The extension has been in use on the spatial data algorithms course at Helsinki University of Technology for two years. The learning results of students using the system are comparable to previous results on a basic data structures and algorithms course using either TRAKLA2 or similar pen and paper exercises. The students' attitudes towards the system have been mostly positive.

1 Introduction

Spatial data is data that is located in a multidimensional space [28]. Spatial data is used in numerous disciplines, such as geographic information systems (GIS), computer graphics, robotics, virtual reality, computer–aided design, biology, VLSI design, and many others. Often, especially in the context of GIS and related disciplines, it is assumed to model geographic locations on the Earth's surface, and their associated properties. In this paper, we discuss spatial data in this context. Thus, each spatial data item contains *spatial information* that describes the location of the data item and associated *data* that describes what it represents. *Spatial data algorithms* (SDA) are algorithms designed to process and manipulate spatial data and *spatial data structures* are entities used to store such data.

Spatial data structures are based on regular non–spatial data structures such as arrays and trees, as well as algorithms that manipulate these basic structures. However, the multidimensional nature of spatial data makes them more complex than the basic data structures. This also makes the design and implementation of efficient spatial data algorithms more complex. For example, multidimensional data do not have a unique order to which they can be arranged.

Furthermore, the spatial nature of the data makes teaching spatial data algorithms more challenging than basic non–spatial data structures and algorithms. For example, in order to show both how located data items are related to each other and how they are arranged in a data structure, typically two illustrations are required. One to show the data elements and the space they occupy and another to show the data structures. Consequently, the learner must be able to connect the illustrations in order to comprehend how the data is organized. Whereas in basic data structures the relations between the data elements are distinguishable with just a single picture.

SDA and associated data structures are an integral part of *geoinformatics*, a branch of science where computer science is applied to cartography and geosciences. Geoinformatics is closely related to cartography, and therefore many different kinds of illustrations are used. Maps are the most fundamental way to represent spatial information. Visualization of maps is in itself a large, varied and important field [42]. Furthermore, since spatial data sets typically have many properties for each location, other information visualization methods are also required. For example, multivariate visualization techniques such as parallel coordinate plots or star plots can be used in conjunction with map views.

Software Visualization (SV) is a branch of software engineering that aims to use graphics and animation to illustrate the different aspects of software [43]. SV is typically divided into two subcategories: program visualization (PV) and algorithm visualization (AV). PV is the use of various visual techniques to enhance the human understanding of computer programs [36]. It is typically used to illustrate actual, implemented programs. AV, on the other hand, illustrates abstractions of algorithmic concepts and is independent of any actual algorithm implementation [36]. In this paper, we will concentrate on the latter side of SV.

One of the primary uses for SV is education. Thus, numerous SV systems have been developed for teaching purposes. However, it has been noted that merely introducing SV to teaching does not significantly improve learning results [18]. Visualizations must be used in a way that activates the learners and enables them to construct and refine their knowledge.

One way to activate the learner is to use exercises where interaction with algorithm visualizations is required in solving the problem. For example, the learner could manipulate data structure visualizations in order to simulate the modifications a real algorithm would do to the data structures in question. We call these exercises *visual algorithm simulation exercises* [23, 30]. The exercises incorporate *automatic assessment* that allows the learners to practice without the need for instructor participation. Automatic assessment is a computerized procedure that takes a learner-made solution to an exercise as an input and produces a mark for the solution without any human intervention. This novel approach seems to be a promising alternative for explaining how these hard-to-grasp spatial data algorithms and data structures work.

Previous studies show that TRAKLA2 is an effective teaching tool [27, 31] for basic data structures and algorithms. We have extended the system to include spatial data structures and algorithms as well [34]. The system has been in use at the SDA course at Helsinki University of Technology for two years. The overall reaction from the students has been positive, and they generally wish to have more TRAKLA2 exercises, as the system does not yet cover all the course topics. Furthermore, there is a strong correlation between successfully solving TRAKLA2 exercises and performing well in the final examination.

The rest of this paper is organized as follows. In section 2, we survey related work on software visualization and automatic assessment. Section 3 contains a more detailed description of the TRAKLA2 system. Sections 4–5 contain the research setting and methods as well as results. Section 6 contains discussion about the results and the use of the system. Section 7 concludes the paper.

2 Background and Related Work

Geoinformatics contains elements from both cartography and earth sciences as well as elements from computer science. However, the focus in teaching geoinformatics is often on how to use available tools and techniques instead of how to develop them. Therefore, geoinformatics courses often deal with how to run different types of analyses, such as numerical modeling [21], simulations [7], or exploratory data analysis [3] and how to use standard tools such as the ESRI ArcGIS environment or Matlab [9]. Internet and various eLearning environments have also been utilized [9, 37].

Information visualization is often used in teaching geoinformatics as well, but the use of software visualization is rare. Several visualization systems exist that can be used to illustrate some spatial data structures, algorithms, and concepts such as geometric algorithms [15, 16, 41] or Voronoi diagrams [12]. It seems, however, that such systems are typically designed for and used in the computer science domain as opposed to geoinformatics. Furthermore, none of the systems come even close to covering the basics of spatial data structures typically used in geoinformatics. In addition, the visualization of spatial structures and algorithms does not currently seem to be an active area of research. A recent paper on the MAVIS algorithm visualization tool has one spatial algorithm example [22], but we are not aware of other recent progress in the field.

In computing education, visualization is utilized more often. Especially for the topic of basic data structures and algorithms, numerous visualization systems have been implemented in many different institutions [1, 17, 20, 38]. Early systems focused on creating clear and attractive visualizations. This is reflected in the work of Price et al., which concentrated on how well different aspects of the graphical representation have been implemented and how versatile the graphical representations are [36]. More recent work recognized that the pedagogical effectiveness of a visualization depends more on the level of engagement the learners have with the visualizations than on the attractiveness of the graphical representation [18]. Therefore, many modern systems are more concerned about the type of interaction they offer than how visually pleasing the graphical representations are. Often the types of interaction are categorized using the engagement taxonomy [33].

Software visualization can be used to create interactive exercises, that the students can solve by manipulating the visual elements [19, 30, 44]. When such exercises are combined with automatic assessment, the learners can get feedback on their solutions without human intervention. Automatic assessment is also commonly used in computing education separately from software visualization. Solutions range from multiple-choice question systems [10] to those assessing programming assignments [4, 39] and style [2]. For more in-depth investigation on the use of automatic assessment, see [5].

3 System Description

3.1 Visual Algorithm Simulation Exercises

Visual algorithm simulation [23, 30] is a novel technique for allowing a user to directly manipulate data structures on the screen. Not only does it give the possibility to modify data structure visualizations, but also to execute algorithms for real data structures by using simple GUI operations. The user can *simulate* real algorithms by manipulating elements on the screen. This concept can easily be extended to automatically assessed *visual algorithm simulation exercises* where the grading is based on comparing the student-made simulation sequence to a sequence produced by an actual algorithm.

In *tracing exercises*, for example, the input for the algorithm is randomized for each student and the task is to simulate the algorithm with the given input. Thus, the solution for the exercise is different for each student. The primary method of interaction is clicking and drag-and-dropping data item visualizations, although other basic GUI components, such as buttons and combo boxes, are also used in some exercises. A typical interface operations in an exercise is to drag an item from its current location to another. This simulates the action of deleting a data element from one structure and inserting it into another. The new location could be another position in the same structure or in another data structure altogether.

In tracing exercises, the input of the algorithm fully determines the control flow of the algorithm's execution, and the task is to reproduce the execution by appropriately manipulating visualizations. At each step, the learner traces the pseudo code and is forced to think of how the algorithm works and carry out the next operation. For example, the learner might drag an item from a stack visualization onto a visualization of a search tree and drop it there, thus inserting the item into the tree. In essence, they are constructing an animation of the algorithm's execution. The assessment of the solution is then carried out by comparing their animation sequence to that of generated by running an actual implementation of the algorithm. Feedback is given in terms of the number of correct steps from the beginning of the sequence and by showing the correct solution as an algorithm animation.

Open tracing exercises, on the other hand, are more exploratory in nature and are used to allow students to examine a specific concept through visual exploration. In these exercises, the correctness of the solution is evaluated based on the end state. An example of such an exercise is the coloring of a red black tree: the student is not required to follow a *specific* algorithm, but is asked to give any correct coloring. Thus, the focus in this exercise is on the *conceptual knowledge* and understanding of the rules that govern the creation of red black trees (such as "the root node is black" or "red nodes cannot have red children"). These conditions are easy to check, and the feedback can explicitly state which constraints are satisfied and which are not. For more discussion about different types of exercises, see [24].

The visualizations in the algorithm simulation exercises can be based on canonical views similar to those in textbooks. Basically all the data structure visualizations are constructed from combinations of *elementary structures* such as array, linked list, trees, and graphs. The *actual data structures* (e.g., binary heap) needed in the exercises are typically derived from the elementary structures.

The data structures can be *presented* on several levels of abstraction depending on the purpose of the visualization. For example, a binary heap is a priority queue ADT (Abstract Data Type) that is implemented as an array. It also has a well known representation in which it is visualized as a binary tree.

In the following, we define a data visualization abstraction model. We show that only a small number of elementary structures are required in order to implement *any* data structure. Furthermore, elementary structures all have widely-accepted visualizations, or *canonical views* that can be used to illustrate any data structure defined in terms of elementary structures.

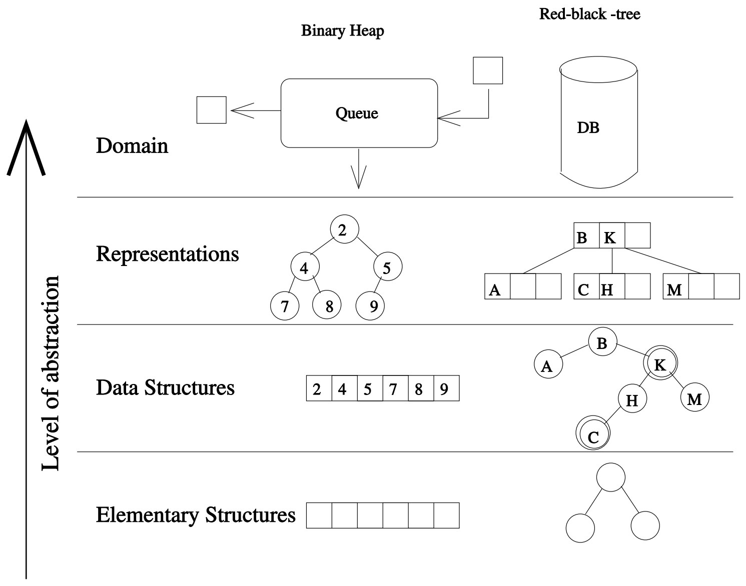


Figure 1: Different levels of Visualization

Let us start with an example illustrated in Figure 1, which shows data structures visualized on four different levels of abstraction. There are two example structures shown in the figure, a binary heap on the left and a red-black-tree on the right. First we will focus on the visualization of the binary heap.

A binary heap is typically implemented using an array. The canonical view of an array is shown on the lowest level in the figure. An array can be used to implement a binary heap by imposing a set of semantics to it: the child nodes of the node i are in positions $2i$ and $2i+1$, and for each node the heap property “the parent is smaller than or equal to the children" holds. A binary heap is shown on the second-lowest level in Figure 1 (data structure view). On this level, the heap is visualized as an array, and therefore the visualization accurately reflects how the data structure is implemented. However, in the array visualization, it is difficult to see whether the heap property holds. The heap property is easier to verify when the data structure is shown using the binary tree representation (representation view). Finally, a binary heap can be utilized, for example, in a network router simulation (domain view). Different packets can have different priorities, and the priority queue could be abstracted into a black box where the internal structure is not shown at all. The packets go in from the right, and forwarded packets come out from the left. If packets are dropped, they are shown below the visualization.

3.1.1 Elementary Structure Level

In general, an elementary structure is a generic data type (a skeleton) that is used to implement data structures. For example, arrays, linked lists, trees, and graphs are such elementary structures. These are archetypes, *reusable basic building blocks* for creating structures, and all the data structures used in textbooks can be implemented in terms of these archetypes. For example, an adjacency list is a composition of an array and linked lists.

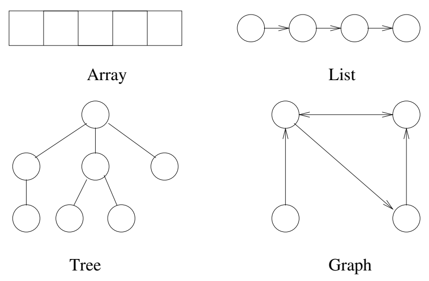


Figure 2: Canonical views

The canonical views of the basic structures are commonly accepted and used, for example, in most textbooks. Anyone familiar with data structures will immediately identify these visualizations as a certain type of structure. Examples of the canonical views can be seen in Figure 2.

The canonical view of an array is a set of boxes arranged side-by-side, with each box containing a single data item. Similarly, the canonical view of a list is a group of nodes arranged in a line, where each node has a reference (line with possibly an arrow at the end) to its successor. The first node of the list is the leftmost one. The canonical view of a (rooted) tree is a number of nodes arranged to levels in a top-down fashion, where each node has a reference to its children, and all nodes that are at an equal distance from the root node are on the same level. The canonical view of a graph is a group of nodes, each of which has references drawn to its successors.

As we can see, elementary structures do not have semantics. The definition of what is an array does not dictate the data types nor the order of the items within. Elementary structures are a kind of archetypes for all such data structures. Similarly, we need to illustrate the actual data and other constraints in order to come up with a meaningful illustration. The first fundamental idea behind elementary structures is the ability to reuse them in representing many different types of data structures. For example, an array can be the basic building block for both a hash table and a binary heap. Actually, we can have a single reusable representation for all data structures that utilize arrays. The second fundamental idea is that only a handful of different elementary structures are needed to implement any data structure. Consequently, only a handful of representations are needed to visualize any data structure.

3.1.2 Data Structure Level

Data structures are implementations of *conceptual models* that encapsulate particular ADTs. This conceptual model defines the elementary structures needed to implement the model as well as the type constraints that lay the foundation for the implementation. The respective data structure is an implementation that also defines the operations needed to modify the structure. For example, a red-black tree is a data structure that has the basic structure of a binary tree, and implements an ADT called the dictionary (operations search, insert, and delete). An example tree is illustrated in Figure 1 (red nodes are shown with double circles).

On data structure level, the visualizations accurately reflect the actual *physical*[1] implementation of the data structure, and show all (relevant) parts of the structure. In the visualization, we are most interested in the layout and the question whether it satisfies the constraints of the data structure. The concept of rotation in red-black trees, for example, can be expressed in terms of changes in this layout (by moving nodes and arcs between nodes).

The layout of the visualization can be customized in order to give the viewer a visual hint of what kind of data structure is being visualized, or to show some additional information about the different parts of the structure. For example, the nodes of a red-black tree are typically drawn either red or black.

3.1.3 Representation Level

A single data structure can often be presented in several ways. In our definition, the representation level contains visualizations that do not necessarily reflect the actual physical implementation, but still show all the data stored in the structure. For example, a binary heap can be presented as an array, which reflects its actual implementation, or as a binary tree that better illustrates the *logical* structure of the heap. Therefore, different representations form a level of abstraction where the visualizations can hide the "physical" characteristics of the data structure in order to make some implicit characteristics more explicit.

On representation level, the visual metaphor used to illustrate the data structure can be changed in order to bring out the desired characteristic of the structure more clearly. For example, a red-black tree can be used to implement a B-tree [13], and therefore it

[1] As opposed to logical that is the case in the next level.

can be visualized as such as well. In Figure 1, a 2-3-4-tree representation is used for the red-black tree on representation level. Furthermore, on the representation level, some visualizations no longer conform to any of the canonical views illustrated in Figure 2. It should be noted, however, that any data structure *can* be visualized using canonical views. Sometimes, however, custom visualizations are better for bringing out specific aspects of the structure.

Alternate visualizations are especially useful when the data stored in the structure is multidimensional. In the elementary structures, there is only a single dimension that is used to show the relationships between the data elements: the visualization of the data key values. If the elements are, for example, points in two-dimensional space, it is difficult for the viewer to grasp the relationships among the elements in a one-dimensional visualization. Whereas if the points are drawn onto a two-dimensional plane, we can more conveniently see the relationships (e.g., distance, direction) between different data points.

3.1.4 Domain Level

Visual metaphors, where the information in the data structure is not fully visible, are considered to be on the top level of abstraction. Typically on this level, we illustrate the data in domain-specific ways that omit most or all of the data structures' details. For example, B-trees are often used as database indexes or storage structures as illustrated by the database symbol in Figure 1. This view hides all the details of the implementation. Therefore, on the domain level, the data structure visualization is analogous to the definition of an ADT. The interface is defined, but the implementation is hidden.

3.1.5 Combined Levels

Often it is useful to show different views of the same data structure simultaneously. For example, binary heap animations are typically based on a binary tree visualization to show how new elements are added or removed. However, having both an array and a binary tree visualization simultaneously to emphasize the fact that its implementation is an array can help the viewer to understand how the logical and physical structure of the heap relate to each other.

It does not matter on how high a level in Figure 1 the representation is, it can be reduced to elementary structures. For example, in GIS applications, the typical representation is a map. In combined level visualizations maps and the corresponding data structures implementing them are show simultaneously.

3.2 TRAKLA2

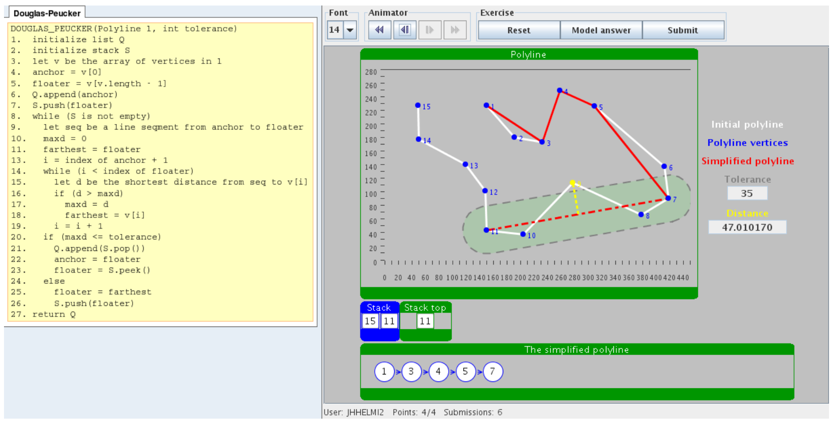
TRAKLA2 is a web-based learning environment aimed at teaching data structures and algorithms [30]. The system is based on an application framework for delivering automatically assessed interactive visual algorithm simulation exercises. The basic exercise set includes assignments for basic data structures, sorting, hashing, graph algorithms, and algorithm analysis. For going beyond the basic data structures and algorithms of core computer science, the system was extended to support the visualization and manipulation of spatial primitives and data structures.

TRAKLA2 is built on top of Matrix, a general-purpose application framework written in Java for creating algorithm animations and simulations. In Matrix, all data structures are visualized by combining instances and variations of the canonical views discussed in the previous section. Moreover, the framework allows for many simultaneous synchronized views of the same underlying data. In other words, we may, for example, have both an array and a tree visualization of a heap data structure where invoking operations via either of them modifies the same underlying data structure and triggers an update of the visualizations. Furthermore, visualizations can be combined hierarchically. For example, a B-tree may be visualized as a tree in which each node contains a data item that is visualized as an array. On the source code level, providing a specific view for a structure in Matrix simply means implementing the respective visualization interface.

The previously existing visualizations in TRAKLA2 exercises can accurately depict the internal hierarchy of a structure, that is, how the data elements are related to each other. For example, a tree is visualized as a layered hierarchy of nodes that are connected by edges that represent links between the data items. However, because of such additional properties as proximity, size, and direction, these types of visualizations cannot adequately represent the additional relationships between spatial data elements. Although, we might use a list of coordinate values to represent spatial data, this type of rudimentary visualization is insufficient for conveying spatial information. For example, tuple of the coordinates of its corners could well represent the bounding rectangle of a node in an R-tree, but it would be hard to make any observations on the overlap or hierarchical composition of the bounding rectangles in the tree. To tackle this, we extended TRAKLA2 for the SDA exercises with a representation level visualization called the *area visualization*.

An area view shows a rectangular region of a 2-dimensional plane. It can be used to represent the properties and relationships of multidimensional vector data in a 2-dimensional coordinate space. For example, in the area visualization, we can show the spatial primitives, such as points, lines, and polygons, that are stored in a spatial data structure. The elements can be differentiated using colors, transparency, line thickness, and dash types. However, while this does allow us to illustrate spatial relationships well, it is equally difficult to depict the structural relationships of data in this view as it is to display spatial attributes with canonical visualizations. Thus, while the area view can be used to provide a good conceptual view of an algorithm, the implementation still operates on actual data structures. To get a complete picture, we generally need two simultaneous views of the data: the data structure level visualization based on canonical views to show how the data is stored and the representation level area view to show how the data elements relate to each other in terms of their spatial attributes. In addition, it is possible, and in some cases useful, to include extra visual cues in the area view that do not explicitly exist in the algorithm or data, but represent conceptual constructs. An example of this is drawing a sweep line in sweep line algorithms.

The set of exercises in TRAKLA2 is divided into rounds with defined publish dates, deadlines, and pass requirements. Figure 3 shows a screenshot of an exercise. It consists of a description and instructions for solving the exercise, pseudo code for the algorithm (in tracing exercises), and an interactive Java applet for carrying out the assignment.



```
DOUGLAS_PEUCKER(Polyline l, int tolerance)
initialize list Q
initialize stack S
let v be the array of vertices in l
anchor = v[0]
floater = v[v.length - 1]
Q.append(anchor)
S.push(floater)
while (S is not empty)
  let seq be a line seqment from anchor to floater
  maxd = 0
  farthest = floater
  i = index of anchor + 1
  while (i < index of floater)
    let d be the shortest distance from seq to v[i]
    if (d > maxd)
      maxd = d
      farthest = v[i]
    i = i + 1
  if (maxd <= tolerance)
    Q.append(S.pop())
    anchor = floater
    floater = S.peek()
  else
    floater = farthest
    S.push(floater)
return Q
```

Figure 3: TRAKLA2 exercise for the Douglas-Peucker line simplification.

For the SDA extension we have implemented two types of exercises as defined in the previous sub-chapter: tracing and open tracing exercises. For example, in an exercise for exploring the construction of a Voronoi diagram by means of the Delaunay triangulation (see Figure 4), the learner is provided with interface operations for modifying the edges of the triangulation and testing for Delaunay conditions. The success is evaluated based on whether the points were correctly connected to create the new triangulation. A list of implemented SDA exercises is in Appendix A.

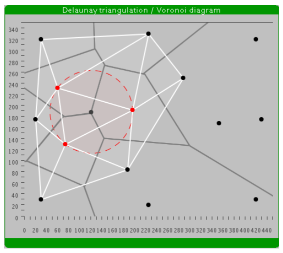


Figure 4: TRAKLA2 exercise for Voronoi diagram construction.

Finally, one of the key ideas in TRAKLA2 exercises is to allow multiple tries for each assignment. This is possible because, for each attempt, a new set of random input is generated. This allows learners to iteratively refine their possibly flawed existing knowledge of an algorithm or concept towards a more viable mental model based on the feedback they receive.

4 Research Setting

The TRAKLA2 SDA extension was implemented for the Spatial Data Algorithms course (SDA course) at Helsinki University of Technology. The course, held on each spring, is aimed at third year geoinformatics students. The prerequisites for the course are basic knowledge on geoinformatics, programming, data structures and algorithms. The SDA course typically has between 15 and 25 students per year. Most have geoinformatics as their major.

Before the TRAKLA2 exercises were added, the SDA course consisted of combined lecture and studio sessions, a programming project, and a final examination (exam). In the lecture and studio sessions, the students work in groups and study spatial algorithms on a conceptual level. Typically, there are 6 or 7 lecture and studio sessions on the course. The programming project, in which the students implement one spatial algorithm, is carried out after the lectures, working either alone or in pairs. Finally, after the lectures have ended, the students have their first chance to take the exam. The final examination typically consists of four or five questions. If they fail at first attempt, the students have several opportunities to retake the exam. The spatial exercises were added to the SDA course in 2007, without modifying any of the other elements on the course.

4.1 Details of the course

In spring 2007, 16 students started the course and 10 of them participated in the first exam. The TRAKLA2 system included 9 spatial exercises. In spring 2008, 20 students started the course and 16 of them participated in the first exam. This time the system included 10 spatial exercises. Seven of the exercises were the same as on spring 2007 course, one exercise was heavily modified between the courses, one was dropped, and two new exercises were added.

In spring 2007, the course was given by a professor who had been in charge of the course for several years. Our plan was to have her be in charge of the course also in spring 2008. Unfortunately, she was able to give only half of the lectures in 2008 due to other obligations. Therefore, half of the lectures in spring 2008 were given by other people. One of the authors gave two of the six lectures, and a visiting lecturer gave one. Course contents were the same in both years.

In spring 2007, the spatial exercises were divided into five rounds, each round consisting of 1–3 exercises. In spring 2008, there were four rounds. In order to pass the course, the students needed to gain at least 50% of the TRAKLA2 points available. A student could submit each exercise as many times as he or she wanted, and the students were not penalized for submitting the exercises late.

The research setting was therefore far from ideal. The content of the course was kept the same after the introduction of TRAKLA2. However, due to the small number of students on the course, we were unable to divide them into different groups in order to compare TRAKLA2 to other teaching methods. Furthermore, since the system was further developed between 2007 and 2008, the students did not receive the same set of exercises on both years. Moreover, in spring 2008, the course was lectured by different people than the previous year. Several independent variables changed and therefore, the two courses are not entirely comparable.

4.2 Data Collected

The data collected from both courses can be divided into two categories: the students' learning outcomes and their attitudes towards the TRAKLA2 system. The data for students' learning outcomes consists of their TRAKLA2 and exam results. The system stores all student submissions, enabling us to see how many times each student has submitted each exercise, and how each submission was assessed. In addition, the students' exam answers were examined.

Two methods were used for collecting data on student attitudes. First, at the end of the course, all students filled out an anonymous course feedback form. This consisted of questions both about the students' attitudes towards the course as a whole and towards the TRAKLA2 system. Second, after the spring 2008 course, four students were interviewed about the system. The students were one Finnish female, one Finnish male, one foreigner female and one foreigner male, aged between 22 and 28 years. The interviews were conducted by the two authors not involved in teaching the course.

5 Results

5.1 Learning Results

The details of the 2007 and 2008 courses are shown in Table 1. The table shows how many students started the course each year, how many of them participated in the final exam, how many TRAKLA2 exercises there were on the course (and how many of those were spatial data algorithm exercises), the total number of TRAKLA2 submissions on the course, and the students' average score. The number of students on the course includes only those who actually participated in some learning activities on the course. In both years, there were students who enrolled for the course but did not do anything. In year 2007, there were five of them, but in 2008 only one.

Table 1: Basic course data for the Spatial Data Algorithm course

year	**# students**	**# in exam**	**# exer. (SDA)**	**# subs**	**avg. score**
2007	16	10	15 (9 SDA)	723	67%
2008	20	16	16 (10 SDA)	1036	83%

It can be seen in Table 1 that in the 2008 course, the students made more submissions and gained a better average score than in 2007 (45.2 submissions on average in 2007 and 51.8 in 2008). However, in 2008 there was one exercise more than in 2007, thus the number of submissions per student per exercise was similar on both courses (3.0 in 2007 and 3.2 in 2008).

The most likely reason for the better scores are the improvements to TRAKLA2 between the two course instances. That is, one exercise was completely redesigned, one exercise was removed, and two new exercises were added to the course. In 2007, the students gained the lowest average scores exactly on the exercises that were redesigned (56%) or removed (57%) before the 2008 course. In addition, in 2008, students gained much better scores on the redesigned exercise (average score 80%), and extremely good scores on the new exercises (average score 98% in both). The affected exercises were worth a total of 16 points in each course. However, the maximum points in 2007 were higher (88 compared with 72 in 2008). Therefore, the affected exercises were proportionally worth a bit more in 2008.

5.1.1 Quantitative Results

Linear regression analysis was used to ascertain whether the students' TRAKLA2 performance was a good indicator of their exam results. The analysis was made both between overall TRAKLA2 and exam results as well as between TRAKLA2 and exam results covering only R-trees [14]. In both years, there were two TRAKLA2 exercises about R-trees. First, the students were required to insert data items into an R-tree, and second, they needed to search for specific data elements in an R-tree. Both years' exams contained one R-tree question.

When the students' exam answers were examined in detail, it was noted that the evaluation of the R-tree exercises differed significantly between the two years. Therefore, in order to make the statistical analysis of the different exam results comparable, the R-tree exercises were independently regraded by two of the authors. Linearly weighted Cohen's Kappa [6] was used to measure the inter-rater reliability since standard Cohen's Kappa measures just how many ratings both raters have put in the same category. In the linearly weighted model the distance between the two ratings is also taken into account. In our opinion this better reflects the fact that if one rater gives a grade 4 and the other a grade 5, the two ratings are still quite close to each other. The agreement between the two gradings was substantial (Kappa 0.64), and the average of the two was used in the linear regression analysis.

Table 2 contains the results of the regression analysis. The table is divided into three categories. First category contains the course information, mainly the year and the number of participants in the exam. The second category contains the results of the linear regression analysis for the whole course, while the third category contains the results for the R-tree exercises. For the linear regression analyses ρ (correlation), adjusted R^2 (strength of relationship) and p (statistical significance) are reported.

Table 2: Learning results on the spatial data algorithms course

Course info		**Whole exam**			**R-trees**		
Year	N	ρ	adj. R^2	p	ρ	adj. R^2	p
2007	10	0.74	0.50	0.01	0.90	0.78	< 0.01
2008	16	0.48	0.18	0.058	0.55	0.25	0.03

As can be seen in table 2, there was a strong correlation between students' performance in TRAKLA2 and the course exams. The correlation is very significant in the R-tree exercises. Also, with the exception of the overall 2008 exam results, the relationship is statistically significant ($p < 0.05$), and even the relationship in 2008 is almost significant ($p = 0.058$).

Similar results have been observed on basic data structures and algorithms courses [25]. In that study, TRAKLA2 exercises were compared with similar exercises done with pen and paper. The study showed that the learning methods gave similar results. Due to the small class sizes on the SDA course, a similar research setup was not possible.

5.1.2 Qualitative Results

Content analysis [26] was used on the students' exam papers in order to investigate two questions. First, how students' experiences with the TRAKLA2 exercises are reflected in their exam answers. Second, what misconceptions students have about the data structures and algorithms asked about in the exams. In the analysis, the answer to each question in a student's exam paper was considered separately, and the analysis was mainly done per question and not per student. All exam questions were considered in the analysis, but most effort was spent on the R-tree questions. R-tree was the only data structure or algorithm that was a part of both years' exam.

Depending on the exam question, we were able to see none, slight or even rather extensive evidence of the effects of TRAKLA2 exercises. The effect was primarily observed in either the diagrams a student drew as part of their answer, or on which parts of a given data structure or algorithm their answers concentrated. However, in the R-tree exercises, there is not much evidence that TRAKLA2 had any direct influence on students' exam answers, especially in the 2008 exam.

In the 2007 exam, many students did not remember R-tree at all, or remembered it completely incorrectly, got 0 points from the exam question. Those who had some recollection of the structure typically drew R-trees where each non-leaf node had either two or three children, similar to the TRAKLA2 exercise. There was no other possible direct influence from TRAKLA2 observed. Average student score, using the R-tree regrading done by the authors, was 1.4 points out of the maximum of 6. In that year, the students' learning material on R-trees included only the lecture notes and the TRAKLA2 exercises. In the lecture notes, however, there were no pictures of R-trees, and therefore the students only saw visualizations in the TRAKLA2 system.

In 2008, the lecture material on R-trees was redesigned and relevant parts from [45] and [40] were included as additional teaching material. The redesigned lecture material also contained R-tree pictures created using TRAKLA2. In 2008, the students' average score was significantly better than in the previous year: 3.75 out of the maximum of 6. Furthermore, the students' answers showed practically no direct influence from TRAKLA2 exercises. The students' diagrams typically were very close to the ones included in [45], where each non-leaf node had two child nodes and each leaf node could contain two data items.

There were, however, exam questions where the influence of the TRAKLA2 exercises could clearly be seen. In the 2007 exam, one question was to describe algorithms for creating a Delaunay triangulation. In the course material, there was one Delaunay triangulation algorithm which had a TRAKLA2 exercise, the expanding wave. Other learning material for this algorithm included the lecture slides and the original research paper [32]. Seven of the ten students described the expanding wave in their answers, and five of them used illustrations. Four out of five used illustrations that were very similar to the visualizations used in the corresponding TRAKLA2 exercise, while one used an illustration similar to those found in [32]. One student actually illustrated the whole algorithm step-by-step using visualizations functionally identical to the ones used in the TRAKLA2 exercise.

In the 2008 exam, one question was about different algorithms for finding the shortest path in various situations. One of the algorithms discussed on the course was about using a visibility graph to find the shortest path in free space with polygon obstacles. The learning material for this consisted of lecture slides, relevant parts of [8], and a TRAKLA2 exercise concentrating on a single rotation of the rotational sweep algorithm. In the exam, students who described the visibility graph almost exclusively concentrated on describing how the algorithm finds the points visible from a given point. Other parts of the algorithm that construct the visibility graph were glossed over by most students. Those who drew pictures, however, typically included the whole visibility graph even if they did not describe it in the text. Only three students included the rotational sweep algorithm in the pictures they drew.

Student misconceptions were also found in the content analysis. The R-tree exercise, especially in the 2008 exam, contained several, for example. The most common ones were that R-tree nodes can have a maximum of two children, and that an R-tree is created by recursively dividing the given area into two subareas covered by the node's children. Most likely source for the first misconceptions is [45], where the example R-tree has two children per node. The source of the second misconception is less clear, but it may be related to the fact that during a split, an R-tree node is divided into two new nodes. Another possible reason is that R-tree was taught after the students had learned about quad-trees, where the area is recursively divided into pieces during construction of the data structure.

There was a consistent, identifiable misconception in one exam question that did not have a corresponding TRAKLA2 exercise. When asked to describe an automatic label placement algorithm [11], most students concentrated on describing how to calculate the fitness of labels for one feature. They did not take into account at all how the labellings of different features may interfere with each other, and how simulated annealing is used to find a good labeling for a set of features. Instead, most students claimed that after the fitness of each potential labeling for a single feature has been calculated, the most fit label position is selected, and failed to mention how other features affect the selection of label position. Other questions did not have any misconceptions that occurred more than once or twice in the whole set of answers.

5.2 Student Attitudes

The students' attitudes to the TRAKLA2 system were primarily measured in two ways: using a course feedback questionnaire and through interviews. Some hints about student attitudes could also be found in the students' TRAKLA2 data.

The **course feedback questionnaire** was given almost identically in both years. The only difference between years was that in one series of questions, where the students were to grade different teaching methods on the course, the scale was changed. In 2007, the scale was from 1 to 4, but in 2008 it was changed to conform to the 0 to 5 scale used to grade exams and courses. The change was done in order to make it easier for the students to answer to the question since they had a direct analogy in the exam grades.

In the questionnaire, there were several questions regarding the system. One was to give it a grade, while other questions discussed the system's usefulness (on a scale from 1–4 where 1 was "not useful" and 4 "very useful"), and the various aspects of TRAKLA2 from pseudo code to the applet's user interface.

In 2007, TRAKLA2 was among the lowest-ranked teaching methods on the course and the average grade given by students was 65% . Most students regarded TRAKLA2 as either a somewhat useful or a quite useful learning method and were of the opinion that it helped their learning. All students regarded TRAKLA2 exercises as suitably challenging. However, many students expressed the opinion that TRAKLA2 was their least favourite part of the course.

In 2008, TRAKLA2 was the highest-ranked teaching method on the course and the average grade rose to 75%. Most students regarded TRAKLA2 exercises as either quite useful or very useful for learning, and were of the opinion that the system helped their learning. All students also thought that the exercises were suitably challenging. This year, no student regarded TRAKLA2 as their least favourite part of the course, and a few actually named it the part of the course they enjoyed the most.

The **interviews** revealed some pros and cons of the system. The problems were mostly related to usability issues while the positive comments were related to the better learning experience. A more thorough report on the interviews is published elsewhere [35]. In the following, however, we sum up the findings.

All of the interviewees criticized the GUI, and the feedback received from the exercises. Even though the GUI is quite simple, some students felt that it is too difficult to learn quickly. They especially complained about how the GUI changed between exercises. In addition, as the feedback is merely a number of correct steps in the simulation sequence, the students hoped for more detailed feedback on the mistakes they made. Moreover, a mistake early on results in lost points, thus a more humane grading scheme would have been appreciated.

Most of the comments were positive, however. Especially the model answer was appreciated as well as the animations and visualizations in general. The students' subjective opinion of the system was that it helped them to learn and memorize the topic easier compared to other teaching methods and learning materials. With the system, the students can actually practice the algorithm, which makes it easier to remember it later. In addition, there is a certain analogy between this and *learning by doing*.

The **students' TRAKLA2 data** shows that many students continued to solve the exercises even after they had gained points required for passing the course. The students needed to gain at least 50% of the TRAKLA2 score in order to pass the course, and did not get any further benefit from getting more points. However, as can be seen from Table 1 the average student score is much higher than 50%. Especially in the 2008 course, students seemed to solve many more exercises than necessary to pass the course. Similar behaviour has also been observed on basic data structures and algorithms courses [29]. It seems that TRAKLA2 inspires students to do extra work also on the SDA course.

6 Discussion

In the two years that TRAKLA2 has been used on the spatial data algorithms course, we have observed several factors that affect students' performance. In order to make a learning environment, such as TRAKLA2, a useful part of the course, many details need to be taken into account. Some of them are specific to the use of spatial algorithms, but many can be applied to other topics as well.

6.1 Conformance with other learning material

One important aspect of integrating a learning environment to a course is ensuring consistency with other learning material. With TRAKLA2, most important are the visualizations used in the exercises. They should be similar enough to those used in other learning material that students can make the connection between the two without too much mental effort. If the visualizations in two pieces of learning material differ too much from each other, the students are likely to either not understand the connection at all, or to favor one of the materials and disregard the other.

A good example of students favoring one visualization over another can be found in the R-tree exam answers. In the 2007 exam answers, when the TRAKLA2 exercises had the only R-tree visualizations the students saw, most of the illustrations in the answers resembled the visualizations used in the system. In 2008, when other learning material on R-trees was added, the influence of TRAKLA2 disappeared from the exam answers. The students clearly preferred the visualizations in the other learning material over the TRAKLA2 exercises.

The most likely cause for preferring the other material is twofold. First, before the final exam, most students tend to review a topic using written learning material instead of redoing already solved TRAKLA2 exercises. Therefore the other learning material is fresher in the students' memory than the exercises. Second, the visualizations used in the TRAKLA2 R-tree exercises were rather different from the other learning material on the topic, and the user interface was designed such that the students' attention was not focused on the area visualizations at all.

In the R-tree examples given in [45, 40], the data items stored in the R-trees are clearly separate from each other, and do not overlap or touch. In the TRAKLA2 R-tree exercise, on the other hand, the input is a polygon network where the data items (polygons) touch each other by definition. Therefore the area visualization of the R-tree in the TRAKLA2 exercise is filled with the data items. This, in turn, hinders, or may actually prevent the student from observing how the nodes of the R-tree are related to each other, and therefore make it harder to grasp how the R-tree is structured. Furthermore, the R-tree is modified by manipulating a tree visualization. In order to solve the exercise, the student does not need to use, or even view, the area visualization at all.

6.2 Clarity of the user interface

In addition to demonstrating how important conformance with other learning material is, the R-tree exercise is also an example of how care must be given to the design and implementation of the user interface. In the R-tree interface, the students do not need to use the area visualization to solve the exercise. This, in turn, makes it possible to completely ignore the spatial nature of the data and focus on just the tree properties of the structure.

This is only one of the problems that can come up with the user interface. In the case of spatial exercises, perhaps the biggest user interface challenge is trying to make the interface such that the exercises can be solved without having to use too much effort to learn the UI. On the data structure level, the interaction boils down to moving and copying data elements, which simply maps to drag-and-dropping them. Whereas, when interacting with data items in the area view, the intended effect is exercise-specific. Data processing needs vary greatly between algorithms and it is difficult to come up with a general-purpose metaphor for mapping GUI interactions to operations on structures in such a way that it is not too restrictive.

We could, for example, bind all operations behind a context-sensitive popup-menu. This would, however, be a very cumbersome interface compared to simply clicking and dragging objects as done in TRAKLA2. Therefore, we must overload the respective interface functionality and give it different meaning in different exercises. For example, in the open tracing Delaunay triangulation exercises, edges can be created by successively clicking two vertices, whereas in the exercise on the expanding wave method of Delaunay triangulation construction, there are two modes of interaction: connecting vertices or computing angles.

The overloading has a clear effect on the students' attitude towards the system, as seen in the interviews. While students like using the system, they dislike having to put so much time and effort into learning how to manipulate the data structures in each exercise.

Even small changes in the system may be significant from the students' point of view. This is seen in one exercise, which was completely redesigned between the two years. The exercise was about the line sweep method for finding line segment intersections. In 2007 the sweep line in the exercise advanced from left to right, and the visualization did not include a representation for it. Also, the visualizations of line segment endpoints were rather large and a linked list was used to store the neighbor relationships of the line segments. In other learning material the sweep line advanced top-down and was shown in the illustrations.

For the 2008 course the visualization was modified to include the sweep line, the line endpoint visualizations were made considerably smaller, the sweep line was modified to advance top-down and line segment neighbour relationships were stored in an array. In effect, all the modifications just changed the visualizations used, leaving the user interface actions almost the same. Despite this, the students' average score for the exercise increased by approximately 25%. Therefore, care should be taken when designing the visualizations.

Another problem with spatial exercises is how to arrange the user interface in such a manner that the students are required to focus on all the visualizations used, and how to make the area visualizations clear and understandable in all situations that might come up. In some cases the area visualization does not always give very useful information to the viewer. For example, if all the elements stored in an R-tree touch each other the area view can quickly become very cluttered and it is hard to distinguish which areas of the visualization are covered by which node, and what are the contents of a given node.

6.3 Software bugs and student attitudes

The presence of software bugs visible to users is another important factor in students' attitudes towards learning environments. In the 2007 course, when the spatial exercises were first taken into use, the system still had a number of bugs that had not been found. These bugs, when found by students, caused a lot of resentment and affected the students' attitude towards the system as a whole.

The bugs were corrected as soon as possible after they were found, but they still caused considerable resentment towards the system. Therefore, even though the number of problems encountered steadily decreased, the students' general attitude towards the system was already set. Their first impression was rather negative, and continued to be such through the course. The bugs were fixed before the 2008 course started and the students' general attitude towards the system was much more positive that year. The software bugs in a system can therefore be decrease the students' motivation a great deal.

6.4 Usability of the system as a learning aid

As seen in Table 2, there is a strong correlation between a student's TRAKLA2 results and their final examination results. Therefore students who know how to solve TRAKLA2 exercises are the ones who typically also do well in the examination. Of course, this does not necessarily mean that TRAKLA2 is a good learning tool. However, the results have been similar when the use of the system has been compared to doing similar exercises done with pen and paper [25]. This, in addition to the students' positive attitude towards the TRAKLA2 system indicates that the system is a viable learning tool. Even if the students might not directly learn much new by doing the exercises, they gain experience with the algorithms and can see whether their knowledge is correct. Furthermore, the students' attitude towards the system – now with the initial annoyances fixed – is very positive. In the interviews the students expressed a desire for more TRAKLA2 exercises.

There are also several ways the system could be improved. Including new exercises in order to cover more spatial data algorithms is one way. Several of the the currently implemented exercises could also use some improvement, although not necessarily as radical as what was done to the line sweep exercise. For example, the R-tree insertion exercise could be modified to better conform with other learning material simply by changing the input. If the polygon network was replaced by a set of polygons that do not touch each other, the exercise might already become much easier to understand. However, even if this is done there is still the problem that in order to solve the exercise the student does not need to view the area visualizations at all.

There are also some algorithms for which we have not been able to design a good exercise. For example, the exercise removed from the 2008 course was about a divide-and-conquer -based algorithm for finding the closest pair of points. The implementation was, however, so clumsy that learning to use the exercise took a long time for the students. Also, the user interface contained so many different things that the student needed to do, that we came to the conclusion that even if a student was able to solve the exercise, their focus had been in winning the exercise and not in understanding the algorithm. Furthermore, in order to keep the number of steps in the exercise reasonable the input for the exercise needed to be very small. Typically with such input, it would have been better to just use the brute-force algorithm for finding the closest pair.

In the field of spatial data algorithms there is also a whole group of algorithms which we deemed to be unsuitable for TRAKLA2 - at least in the current implementation. Spatial data can be divided into two models: object data model, where spatial elements are stored as distinct objects and field data model, where spatial data is stored as continuous fields. In the object data model the execution of algorithms is typically ruled by the input. With different inputs different lines of the algorithm code are executed in different order. Therefore, when doing tracing exercises, at each step of the algorithm simulation the student needs to think what his next action would be with the given input. With field algorithms, however, the algorithm execution is not governed by the input. The same lines of code are typically executed in the same order, no matter what the input is. Only the numerical variables carried from one command to the next are different. Therefore such algorithms are not, in our opinion, suitable for TRAKLA2. No matter what the exercise was, the algorithm simulation would always be more or less the same, thus making the exercises both boring and repetitive.

7 Conclusions

In this work we have described a spatial data algorithm extension to the TRAKLA2 learning environment. The extension has been used on a real spatial data algorithms course at Helsinki University of Technology for two years.

The system is based on using several different algorithm visualizations at the same time to show both how spatial data elements are related to one another and how they are arranged in data structures. To accomplish this, we have implemented several different visualizations on different levels of abstraction to the TRAKLA2 system.

Initially, the students' attitudes towards the spatial exercises were rather poor, mainly due to a large number of software bugs and complicated UI. After further system development, where the exercises were made easier to use and a number bugs were fixed, the student attitudes towards the system became predominately positive. Therefore, care should be taken when implementing computerized exercises. The exercises should fit with other learning material used on the course, they should be easy to use and engage the user in meaningful ways. In the case of spatial data algorithms the exercises should emphasize the spatial nature of the data.

The learning results gained with the system are similar to those witnessed previously when using TRAKLA2 with basic data structures and algorithms. Therefore using the system appears to be a viable alternative to similar classroom exercises. It also seems that the idea of visual algorithm simulation exercises can be extended to cover SDA.

There are also several promising avenues for future system development. TRAKLA2 does not yet cover the whole SDA course. Also, several existing exercises could be improved by making them clearer and more consistent with the other learning material.

All in all, the TRAKLA2 spatial exercises have proven to be a viable learning method. However, in order to make the most of them, care should be taken in planning and implementing the exercises. Even if an exercise is good by itself, it may not be very effective if it does not fit in well with the other learning material.

References

[1] A. Akingbade, T. Finley, D. Jackson, P. Patel, and S. H. Rodger. JAWAA: easy web-based animation from CS0 to advanced CS courses. In *Proceedings of the 34th SIGCSE technical symposium on Computer science education, SIGCSE'03*, pages 162–166. ACM Press, 2003.

[2] K. Ala-Mutka, T. Uimonen, and H.-M. Järvinen. Supporting students in C++ programming courses with automatic program style assessment. *Journal of Information Technology Education, volume 3*, pages 245–262, 2004.

[3] G. Andrienko, N. Andrienko, R. Fischer, V. Mues, and A. Schuck. Reactions to geovisualization: an experience from a european project. *International Journal of Geographical Information Science*, 20(10):1149 – 1171, 2006.

[4] S. Benford, E. Burke, E. Foxley, N. Gutteridge, and A. M. Zin. Ceilidh: A course administration and marking system. In *Proceedings of the 1st International Conference of Computer Based Learning*, Vienna, Austria, 1993.

[5] J. Carter, J. English, K. Ala-Mutka, M. Dick, W. Fone, U. Fuller, and J. Sheard. ITICSE working group report: How shall we assess this? *SIGCSE Bulletin*, 35(4):107–123, 2003.

[6] J. Cohen. Weighted kappa: Nominal scale agreement with provision for scale and disagreement or partial credit. *Psychological Bulletin*, 70:213–220, 1968.

[7] J. R. Crouch, Y. Shen, J. A. Austin, and M. S. Dinniman. An educational interactive numerical model of the chesapeake bay. *Computers & GeosciencesVolume*, 34(3):247–258, 2008.

[8] M. de Berg, M. V. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer, 2000.

[9] G. Degrande and K. Geraedts. An electronic learning environment for the study of seismic wave propagation. *Computers & GeosciencesVolume*, 24(6):569–591, 2008.

[10] P. Denny, J. Hamer, A. Luxton-Reilly, and H. Purchase. Peerwise: students sharing their multiple choice questions. In *ICER '08: Proceeding of the fourth international workshop on Computing education research*, pages 51–58, New York, NY, USA, 2008. ACM.

[11] S. Edmondson, J. Christensen, J. Marks, and S. Shieber. A general cartographic labeling algorithm. *Cartographica*, 33(4):13–23, 1996.

[12] J. Fisher. Visualizing the connection among convex hull, voronoi diagram and delaunay triangulation. In *37th Midwest Instruction and Computing Symposium*, 2004.

[13] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proceedings of the 19th Annual Symposium on Foundations of Computer Science*, pages 8–21. IEEE Computer Society, 1978.

[14] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[15] A. Hausner and D. P. Dobkin. GAWAIN: visualizing geometric algorithms with web-based animation. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, pages 411–412, New York, NY, USA, 1998. ACM.

[16] C. A. Hipke and S. Schuierer. Vega – a user-centered approach to the distributed visualization of geometric algorithms. Technical report, University of Freiburg, 1998.

[17] C. Hundhausen and J. L. Brown. What you see is what you code: a radically dynamic algorithm visualization development model for n ovice learners. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 163–170, September 2005.

[18] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13(3):259–290, June 2002.

[19] T. Janhunen, T. Jussia, M. Järvisalo, and E. Oikarinen. Teaching smullyan's analytic tableaux in a scalable learning environment. In A. Korhonen and L. Malmi, editors, *Kolin Kolistelut/Koli Calling. Proceedings of the Fourth Finnish/Baltic Sea Conference on Computer Science Education*, pages 85 – 94. Helsinki University of Technology, 2004.

[20] V. Karavirta, A. Korhonen, L. Malmi, and K. Stålnacke. MatrixPro – A tool for on-the-fly demonstration of data structures and algorithms. In *Proceedings of the Third Program Visualization Workshop*, pages 26–33, The University of Warwick, UK, July 2004. Department of Computer Science, University of Warwick, UK.

[21] D. Karssenberg, P. A. Burrough, R. Sluiter, and K. de Jong. The pcraster software and course materials for teaching numerical modelling in the environmental sciences. *Transactions in GIS*, 5(2):99–110, 2001.

[22] I. Koifman, I. Shimshoni, and A. Tal. Mavis: A multi-level algorithm visualization system within a collaborative distance learning environment. *Journal of Visual Languages & Computing*, 19(2):182–202, April 2008.

[23] A. Korhonen. *Visual Algorithm Simulation*. Doctoral dissertation (tech rep. no. tko-a40/03), Helsinki University of Technology, 2003.

[24] A. Korhonen and L. Malmi. Taxonomy of visual algorithm simulation exercises. In *Proceedings of the Third Program Visualization Workshop*, pages 118–125, The University of Warwick, UK, July 2004.

[25] A. Korhonen, L. Malmi, P. Myllyselkä, and P. Scheinin. Does it make a difference if students exercise on the web or in the classroom? In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 121–124, Aarhus, Denmark, 2002. ACM Press, New York.

[26] K. Krippendorff. *Content analysis : an introduction to its methodology*. Thousand Oaks, CA, 2004.

[27] M.-J. Laakso, T. Salakoski, L. Grandell, X. Qiu, A. Korhonen, and L. Malmi. Multi-perspective study of novice learners adopting the visual algorithm simulation exercise system TRAKLA2. *Informatics in Education*, 4(1):49–68, 2005.

[28] R. Laurini and D. Thompson. *Fundamentals of spatial information systems*. Academic Press, 1992.

[29] L. Malmi, V. Karavirta, A. Korhonen, and J. Nikander. Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal of Educational Resources in Computing*, 5(3), September 2005.

[30] L. Malmi, V. Karavirta, A. Korhonen, J. Nikander, O. Seppälä, and P. Silvasti. Visual algorithm simulation exercise system with automatic assessment: TRAKLA2. *Informatics in Education*, 3(2):267–288, 2004.

[31] L. Malmi, A. Korhonen, and R. Saikkonen. Experiences in automatic assessment on mass courses and issues for designing virtual courses. In *Proceedings of The 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'02*, pages 55–59, Aarhus, Denmark, 2002. ACM Press, New York.

[32] M. J. McCullagh and C. G. Ross. Delaunay triangulation of a random data set for isarithmic mapping. *The Cartographic Journal*, 17(2), December 1980.

[33] T. L. Naps, G. Rößling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2003.

[34] J. Nikander and J. Helminen. Algorithm visualization in teaching spatial data algorithms. In *11th International Conference Information Visualization IV2007*, pages 505–510. IEEE Computer Society, July 2007.

[35] J. Nikander, J. Helminen, and A. Korhonen. Experiences on using trakla2 to teach spatial data algorithms. In G. Rössling, editor, *Proceedings of the Fifth Program Visualization Workshop (PVW 2008)*, 2008.

[36] B. A. Price, R. M. Baecker, and I. S. Small. A principled taxonomy of software visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

[37] R. Purves, D. Medyckyj-Scott, and W. Mackaness. The e-mapscholar project–an example of interoperability in giscience education. *Computers & GeosciencesVolume*, 31(2):189–198, 2005.

[38] G. Rößling. ANIMAL-FARM*: An Extensible Framework for Algorithm Visualization*. Phd thesis, University of Siegen, Germany, 2002. Available online at http://www.ub.uni-siegen.de/epub/diss/roessling.htm.

[39] R. Saikkonen, L. Malmi, and A. Korhonen. Fully automatic assessment of programming exercises. In *Proceedings of the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, ITiCSE'01*, pages 133–136, Canterbury, UK, 2001. ACM Press, New York.

[40] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[41] M. Shneerson and A. Tal. Interactive collaborative visualization environment for geometric computing. *Journal of Visual Languages & Computing*, 11(6):615–637, December 2000.

[42] T. A. Slocum, R. B. McMaster, F. C. Kessler, and H. H. Howard. *Thematic Cartography and Geographic Visualization*. Prentice-Hall, 2004.

[43] J. T. Stasko, J. B. Domingue, M. H. Brown, and B. A. Price. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1998.

[44] V. Tscherter, R. Lamprecht, and J. Nievergelt. Exorciser: Automatic generation and interactive grading of exercises in the theory of computation. In *Fourth International Conference on New Educational Environments*, pages 47–50, 2002.

[45] M. Worboys and M. Duckham. *GIS: A Computing Perspective*. Taylor & Francis, 2004.

APPENDIX

Table 3: spatial exercises in the TRAKLA2 system

name	**description**	**Years used**
Point in polygon	The learner is to find whether a point is inside a polygon by counting the number of intersections a half-line drawn from the point has with the polygon.	2007, 2008
Closest pair of points	The learner is to find the closest pair of points in a set of points by using a divide-and-conquer approach based on the merge-sort algorith.	2007
Douglas-Peucker line simplification	The learner is to simplify a polyline using the Douglas-Peucker line simplification algorithm.	2007,2008
Line sweep	The learner is to find line segment intersections in a set of line segmentsby using the line sweep algorithm. The exercise was completely redesigned between the two courses.	2007, 2008
Voronoi construction	The learner is to construct a valid Voronoi diagram from a set of points. There is no need to follow a specific algorithm, only the end result is assessed.	2008
Adding a point to TIN	The learner is to add three new points to a Delaunay triangulation and to modify the triangulation so that it still stays valid. There is no need to follow a specific algorithm, only the end result is assessed.	2008
Expanding wave-method	The learner is to construct a Delaunay triangulation using the expanding wave algorithm.	2007, 2008
Visibility with rotational sweep	The learner is to find polygon end points visible from a given point by using the rotational sweep algorithm.	2007,2008
R-tree insert	The learner is to insert a number polygons into an R-tree.	2007,2008
Point-region quadtree insert	The learner is to insert a number of points into a point-region quadtree.	2007,2008
Point in polygon with R-tree	Point in polygon with the edges of the polygon are in an R-tree and the learner needs to seach the R-tree for edges that may cross the half-line.	2007,2008